ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

RANCANG BANGUN APLIKASI HOSTED CONTINUOUS INTEGRATION DENGAN KEMAMPUAN PENETRATION TESTING UNTUK APLIKASI BERBASIS WEB

I GEDE PUTU SURYA DARMA PUTRA
NRP 5112 100 112

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Baskoro Adi P, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

RANCANG BANGUN APLIKASI HOSTED CONTINUOUS INTEGRATION DENGAN KEMAMPUAN PENETRATION TESTING UNTUK APLIKASI BERBASIS WEB

I GEDE PUTU SURYA DARMA PUTRA
NRP 5112 100 112

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Baskoro Adi P, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

DESIGN AND IMPLEMENTATION OF HOSTED CONTINUOUS INTEGRATION APPLICATION WITH PENETRATION TESTING CAPABILITIES FOR WEB-BASED APPLICATIONS

I GEDE PUTU SURYA DARMA PUTRA
NRP 5112 100 112

Supervisor I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Supervisor II
Baskoro Adi P, S.Kom., M.Kom.

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI HOSTED CONTINUOUS INTEGRATION DENGAN KEMAMPUAN PENETRATION TESTING UNTUK APLIKASI BERBASIS WEB

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
I GEDE PUTU SURYA DARMA PUTRA
NRP: 5112 100 112

Disetujui oleh Dosen Pembimbing Tugas Akhir

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. .....
NIP: 197708242006041001 (Pembimbing 1)

Baskoro Adi P, S.Kom., M.Kom. .....
NIP: 198702182014041001 (Pembimbing 2)

SURABAYA
Juli 2016

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN APLIKASI HOSTED CONTINUOUS INTEGRATION DENGAN KEMAMPUAN PENETRATION TESTING UNTUK APLIKASI BERBASIS WEB

Nama	:	**I GEDE PUTU SURYA DARMA PUTRA**
NRP	:	**5112 100 112**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Pembimbing II	:	**Baskoro Adi P, S.Kom., M.Kom.**

Abstrak

Continuous Integration merupakan suatu konsep yang menawarkan kemampuan integrasi, uji coba fungsionalitas, serta penanaman aplikasi pada lingkungan produksi secara otomatis. Konsep ini memungkinkan kode sumber dari pengembang yang berbeda bisa diintegrasikan, kemudian secara otomatis dilakukan uji fungsionalitas terhadap kode sumber tersebut untuk memastikan kelayakannya. Continuous Integration mendukung pengembangan aplikasi bisa berlangsung dengan cepat dan memastikan aplikasi yang ada pada lingkungan produksi bisa menjalankan fungsionalitasnya dengan baik.

Unit test merupakan istilah yang digunakan pada proses uji fungsionalitas aplikasi dalam Continuous Integration. Keberhasilan dari unit test ditunjukkan dengan keberhasilan perangkat lunak menghasilkan keluaran berdasarkan masukan yang disediakan sesuai batasan yang telah dibuat. Proses ini mengesampingkan aspek keamanan aplikasi karena pada dasarnya unit test hanya bertujuan untuk memastikan aplikasi bisa berfungsi sesuai kebutuhan. Oleh karena itu, pihak

pengembang harus menggunakan alat kakas tambahan untuk melakukan penetration testing dan menganalisa celah keamanan pada aplikasi. Proses penetration testing tersebut berlangsung setelah proses pengembangan selesai. Mengingat aspek keamanan merupakan hal krusial yang perlu diperhatikan pada aplikasi, muncullah sebuah gagasan baru yakni aplikasi Continuous Integration yang mampu melakukan penetration testing. Solusi ini bertujuan untuk memastikan kode sumber tidak memiliki celah keamanan, atau memberikan peringatan dini kepada pengembang agar memperbaiki kode sumber jika memiliki celah keamanan.

Berdasarkan hasil uji coba, sistem yang dibangun pada Tugas Akhir ini mampu menjalankan aktivitas penetration testing secara otomatis maupun terjadwal pada saat proses pengembangan berlangsung serta mampu memberikan notifikasi hasil penetration testing melalui e-mail. Kelemahan yang terdapat pada sistem ini adalah konsumsi memory yang tinggi untuk setiap aktivitas penetration testing yang berlangsung, sehingga diperlukan sumber daya yang besar pula untuk menjalankan aktivitas penetration testing dalam jumlah besar.
***Kata-Kunci**: Continuous Integration, Penetration Testing.*

DESIGN AND IMPLEMENTATION OF HOSTED CONTINUOUS INTEGRATION APPLICATION WITH PENETRATION TESTING CAPABILITIES FOR WEB-BASED APPLICATIONS

Name	:	**I GEDE PUTU SURYA DARMA PUTRA**
NRP	:	**5112 100 112**
Major	:	**Informatics FTIf**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Supervisor II	:	**Baskoro Adi P, S.Kom., M.Kom.**

Abstract

Continuous Integration is a concept that offers automation when doing integration, functional test and deployment process of application. Different source codes from developers within a team are tested to ensure the eligibility, then integrated automatically to produce a good application. Continuous Integration supports rapid development of applications and ensures everything inside production environment works fine.

Unit test is a process of functional testing for application in Continuous Integration system. It will report a success result if an application responds with the correct output for every provided test case based on functional requirements of application. This process override application security aspect because basically unit test aims to ensure the application can run the correct functions. Therefore, developers must take other tools to do penetration testing and analyze the security risks. Penetration test happen after the application out of development phase. Considering the important of security aspect for application, comes a new idea that Continuous Integration

system can do penetration testing. This solution aims to ensure the application has a good level of security or provides the earliest possible warning for developers when the application has security risks.

Based on trial results, the system designed is capable of running penetration test automatically during software development process and is able to provide the result of penetration testing via e-mail. The weakness of this system is high memory consumption for every penetration testing activity, so greater resources are necessary to run large numbers of penetration testing activities.

***Keywords**: Continuous Integration, Penetration Testing.*

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB 1

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dam batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Teknologi pembuatan perangkat lunak berkembang pesat, tidak hanya mengikuti perkembangan kebutuhan konsumen, namun juga mengikuti perkembangan kebutuhan di pihak pengembang. Pesatnya pertumbuhan teknologi menyebabkan permintaan akan perangkat lunak semakin tinggi. Pihak pengembang pun dituntut untuk bisa mengefisienkan waktu pengerjaan, sehingga perangkat lunak bisa cepat diproduksi dan dioperasikan oleh konsumen. Untuk menjawab tantangan-tantangan yang ada, berbagai konsep yang memudahkan berjalannya proses pembuatan dan pengembangan aplikasi hadir sebagai solusi yang bisa digunakan pihak pengembang dalam proses pengerjaan proyek perangkat lunak. *Continuous Integration* merupakan suatu konsep yang menawarkan kemampuan integrasi, uji coba fungsionalitas, serta penanaman aplikasi pada lingkungan produksi secara otomatis. Konsep ini memungkinkan kode sumber dari pengembang yang berbeda bisa diintegrasikan, kemudian secara otomatis dilakukan uji fungsionalitas terhadap kode sumber tersebut untuk memastikan kelayakannya. *Continuous Integration* memastikan bahwa aplikasi yang ada pada lingkungan produksi adalah aplikasi yang bisa menjalankan fungsionalitasnya dengan baik.

Uji fungsionalitas atau biasa disebut dengan istilah *unit test* adalah proses uji coba terhadap fungsi-fungsi aplikasi dengan cara menyediakan batasan masukan dan keluaran. Keberhasilan dari *unit test* ditunjukkan dengan keberhasilan perangkat lunak

menghasilkan keluaran berdasarkan masukan yang disediakan sesuai batasan yang telah dibuat. Namun, terdapat kekurangan pada alat kakas *Continuous Integration* yang sudah ada, yakni *unit test* tidak melakukan uji coba terhadap kemungkinan celah keamanan yang ada pada perangkat lunak. Pihak pengembang harus menggunakan alat kakas tambahan untuk menganalisa celah keamanan, seperti: OWASP ZAP, SQLMap, dan sejenisnya. Berdasarkan kekurangan tersebut, penulis menawarkan sebuah aplikasi *Hosted Continuous Integration* yang mampu melakukan *penetration testing* pada saat proses pengembangan berlangsung. Solusi ini bertujuan untuk memastikan kode sumber tidak memiliki celah keamanan, atau memberikan peringatan dini kepada pengembang agar memperbaiki kode sumber jika memiliki celah keamanan.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat pada Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana membuat aplikasi *Hosted Continuous Integration* yang mampu menjalankan *penetration testing*?
2. Bagaimana melakukan *penetration testing* selama proses pengembangan suatu aplikasi berlangsung, bukan setelah aplikasi melewati proses produksi?
3. Bagaimana cara menginformasikan keberadaan celah keamanaan kepada anggota tim pengembang aplikasi ketika tahap pengembangan berlangsung, sehingga celah tersebut bisa segera diperbaiki?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, diantaranya sebagai berikut:

1. Aplikasi Hosted Continuous Integration yang dibangun merupakan pengembangan dari suatu kerangka kerja Continuous Integration yang sudah ada.
2. Pendeteksian celah keamanan difokuskan pada aplikasi web berbasis PHP.
3. Celah keamanan yang dideteksi adalah SQL Injection dan Cross Site Scripting (XSS), Directory Indexing, Path Traversal, dan Remote File Inclusion.
4. Laporan mengenai hasil uji kelayakan aplikasi oleh aplikasi Continuous Integration yang dibuat ditampilkan pada halaman website serta dikirim melalui e-mail kepada semua kontributor yang terlibat pada proses pengembangan tersebut.
5. Version Control System yang digunakan untuk uji coba sistem adalah GitHub.

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Membuat aplikasi *Hosted Continuous Integration* yang mampu melakukan *penetration testing*.
2. Aplikasi *Hosted Continuous Integration* mampu mendeteksi adanya celah keamanan *SQL Injection*, *Cross Site Scripting (XSS)*, *Directory Indexing*, *Path Traversal*, dan *Remote File Inclusion* pada aplikasi web berbasis PHP selama tahap pengembangan proyek perangkat lunak.
3. Aplikasi *Hosted Continuous Integration* mampu memberikan notifikasi kepada pengembang akan tingkat keamanan aplikasi yang sedang dikembangkan.

1.5 Manfaat

Adapun manfaat dari pengerjaan Tugas Akhir ini, antara lain:

1. Memudahkan tim pengembang aplikasi untuk mengetahui tingkat keamanan aplikasi yang sedang dikembangkan selama proses pengembangan berlangsung.
2. Memudahkan tim pengembang aplikasi dalam memeriksa keberadaan celah keamanan, sebab proses pemeriksaan celah sudah dijalankan otomatis oleh aplikasi *Hosted Continuous Integration*.
3. Adanya pemberitahuan mengenai celah keamanan pada aplikasi memberikan kesempatan bagi tim pengembang untuk menutup celah keamanan yang ada sebelum aplikasi menuju tahap produksi.

1.6 Metodologi

Metodologi yang digunakan pada pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Studi Literatur
   Studi literatur merupakan langkah yang dilakukan untuk mendukung dan memastikan setiap tahap pengerjaan tugas akhir sesuai dengan standar dan konsep yang berlaku. Pada tahap studi literatur ini, akan dilakukan studi mendalam mengenai *Continuous Integration*, *Unit Test*, *Penetration Test*, tipe-tipe serangan dan kemungkinan celah yang ada pada aplikasi berbasi web, serta cara penanganan dan pencegahannya. Adapun literatur yang dijadikan sumber berasal dari paper, buku, materi perkuliahan, forum serta artikel dari internet.
2. Analisis dan Desain Perangkat Lunak
   Pada tahap ini dilakukan beberapa tahapan meliputi perancangan arsitektur aplikasi, serta perancangan

antarmuka aplikasi.

3. Implementasi Perangkat Lunak
   Pada tahap ini dilakukan beberapa tahapan implementasi perangkat lunak yakni, tahap pengerjaan aplikasi *Continuous Integration*, pengerjaan mekanisme untuk *penetration testing*, dan mempersiapkan aplikasi website bercelah untuk uji coba. Alat kakas yang digunakan selama tahap implementasi ini didominasi oleh Sublime Text dan Vim sebagai *text editor* serta Linux Terminal. Adapun bahasa pemrograman yang akan digunakan antara lain: Python sebagai bahasa pemrograman pada aplikasi *Continuous Integration* dan mekanisme untuk *penetration testing*, serta PHP sebagai bahasa pemrograman pada aplikasi website bercelah untuk uji coba. Selain itu, mekanisme untuk *penetration testing* juga didukung oleh alat kakas W3af.
4. Pengujian dan Evaluasi
   Tahap pengujian dan evaluasi dilakukan dengan cara menyambungkan aplikasi dengan repositori GitHub berisi aplikasi website berbasi PHP yang memiliki celah keamanan. Pada repositori tersebut dilakukan beberapa kali perubahan kode program untuk menutup celah keamanan yang dilaporkan, kemudian mengevaluasi perubahan laporan celah kemanan yang dihasilkan oleh aplikasi.

1.7 Sistematika Laporan

Sistematika buku Tugas Akhir ini, antara lain:

1. Bab I, Pendahuluan
   Bab ini membahas latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan Tugas Akhir.

2. Bab II, Tinjauan Pustaka
   Bab ini membahas dasar teori yang berkaitan dengan topik Tugas Akhir.
3. Bab III, Desain dan Perancangan Bab ini membahas perancangan arsitektur dan cara kerja sistem yang akan diimplementasikan.
4. Bab IV, Implementasi
   Bab ini membahas implementasi sistem dari rancangan pada tahap perancangan, meliputi *pseudocode* yang terdapat pada sistem.
5. Bab V, Uji Coba dan Evaluasi
   Bab ini membahas uji coba fungsionalitas sistem dengan cara melihat keluaran yang dihasilkan oleh sistem, analisis dan evaluasi terhadap keberhasilan fungsi serta penggunaan sumber daya selama proses pengoperasian sistem.
6. Bab VI, Penutup
   Bab ini berisi kesimpulan dari hasil uji coba serta saran untuk pengembangan lebih lanjut.

BAB 2

LANDASAN TEORI

2.1 Continuous Integration

Continuous Integration merupakan sebuah praktik dalam pengembangan perangkat lunak yang dilakukan oleh suatu tim. Dalam praktik ini, semua anggota tim melakukan proses integrasi secara berkala terhadap bagian pekerjaan yang dilakukannya dengan bagian pekerjaan yang dilakukan anggota tim lainnya. Proses integrasi tersebut berlangsung pada suatu direktori pusat yang menyimpan aplikasi secara utuh dan nantinya akan menjadi hasil akhir dari proses pengembangan. Tiap anggota tim menambahkan perubahan pada direktori tersebut, sehingga anggota lainnya bisa mendapatkan perubahan yang terjadi pada aplikasi. Setiap perubahan yang terjadi pada direktori pusat akan dilakukan uji coba fungsionalitas, sehingga bisa dipastikan bahwa direktori pusat menyimpan aplikasi yang benar.

Dewasa ini, praktik *Continuous Integration* didukung oleh alat kakas yang memudahkan pengembang aplikasi dalam menyelesaikan pengerjaannya. Alat kakas tersebut disebut dengan *Version Control System* (VCS). Beberapa contoh VCS yang sering digunakan, antara lain: Git, SVN, CVS, dan Mercurial.

2.2 Version Control System

Version Control System (VCS) adalah suatu sistem repositori berkas yang biasanya digunakan dalam pengembangan perangkat lunak. VCS memungkinkan repositori tersebut dipantau perkembangannya, mulai dari perubahan yang terjadi, pengguna yang melakukan perubahan, kapan perubahan terjadi, dan lain sebagainya. Oleh karena kemampuan yang dimilikinya, VCS cocok digunakan pada proses pengembangan perangkat lunak dengan metode kolaborasi. Beberapa contoh alat kakas VCS

yang sering digunakan dalam pengembangan perangkat lunak, antara lain: Git, SVN, CVS, Mercurial, Bazaar, LibreSource, dan Monotone.

2.3 Penetration Testing

Penetration testing atau biasa disebut dengan *pen test* merupakan suatu metode untuk mengevaluasi keamanan dari suatu infrastruktur teknologi informasi dengan cara menyimulasikan serangan-serangan yang mungkin saja bisa terjadi terhadap celah keamanan yang ada pada infrastuktur tersebut. Infrastruktur yang dimaksud mencakup infrastuktur jaringan, sistem operasi, maupun aplikasi yang tertanam di dalamnya. Hasil dari *penetration testing* nantinya bisa digunakan untuk menentukan keputusan pengamanan bagi infrastruktur yang bersangkutan.

2.4 SQL Injection

SQL Injection merupakan teknik serangan terhadap aplikasi dengan cara menyisipkan perintah-perintah SQL pada baris masukan aplikasi. Aplikasi yang tidak memiliki penanganan akan karakter-karakter khusus atau penyaringan masukan akan mudah untuk dieksploitasi menggunakan teknik ini. Melalui teknik ini penyerang bisa mendapatkan data-data yang seharusnya tidak ditampilkan ke pengguna, mendapatkan hak akses tanpa harus melalui proses autentikasi, melakukan manipulasi data, bahkan mengambil alih kendali *administrator* basis data.

2.5 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) merupakan teknik serangan terhadap aplikasi dengan cara menyisipkan baris kode berbahaya,

umumnya dalam bentuk *browser side script*, agar dieksekusi oleh pengguna lain. Teknik serangan ini digunakan oleh penyerang untuk mengambil data-data sensitif pengguna seperti *cookies*, *session token*, dan data-data sensitif lainnya yang tersimpan pada peramban web dan digunakan dalam mengakses suatu *website*. Menurut Acunetix, terdapat tiga jenis serangan XSS, antara lain:

- ***Stored XSS***
  Stored XSS dianggap sebagai serangan XSS paling berbahaya. Pada serangan XSS jenis ini, penyerang menyisipkan kode berbahaya dan membuatnya tersimpan secara permanen pada aplikasi. Contohnya adalah penyisipan kode berbahaya pada kolom komentar suatu website forum. Kode tersebut akan tersimpan dalam basis data aplikasi sebagai komentar, kemudian akan dieksekusi oleh pengguna lain ketika mereka mengakses halaman yang menyertakan komentar tersebut.
- ***Reflected XSS***
  Reflected XSS merupakan teknik serangan XSS yang umum digunakan. Cara kerjanya adalah dengan mengirimkan kode berbahaya bersamaan dengan request kepada web server, kemudian web server akan mengirimkan kembali kode tersebut kepada pengguna sebagai respon atas request yang dijalankan sebelumnya. Ketika respon diterima maka kode berbahaya tersebut akan dieksekusi oleh pengguna. Penyerang biasanya menggunakan media sosial untuk menyebarkan tautan yang mengarahkan pengguna pada suatu website dengan mengikutsertakan kode berbahaya pada request yang dikirimkan, kemudian mengeksekusi kode berbahaya tersebut ketika respon dari web server diterima kembali oleh pengguna.
- ***DOM-based XSS***
  DOM-based XSS merupakan teknik serangan XSS yang

bisa dijalankan ketika aplikasi memiliki client side script yang menuliskan masukan pengguna sebagai isi dari sebuah DOM (Document Object Model). Penyerang akan menyisipkan kode berbahaya pada kolom masukan pengguna, kemudian ketika aplikasi memroses masukan tersebut dan memasukkannya ke sebuah DOM, maka kode berbahaya tersebut akan dieksekusi.

2.6 Directory Indexing

Directory Indexing adalah suatu celah dimana penyerang mampu melihat daftar *file* yang terdapat pada direktori aplikasi. Terbukanya daftar *file* tersebut memungkinkan penyerang mendapatkan informasi-informasi sensitif yang tersimpan pada *file* tertentu.

2.7 Path Traversal

Path Traversal merupakan sebuah serangan yang bisa dilancarkan terhadap aplikasi web dengan cara memanipulasi celah masukan pengguna. Tujuan dari serangan ini adalah untuk mengakses direktori *server* di luar direktori aplikasi yang mungkin saja menjadi tempat penyimpanan informasi sensitif.

2.8 Remote File Inclusion

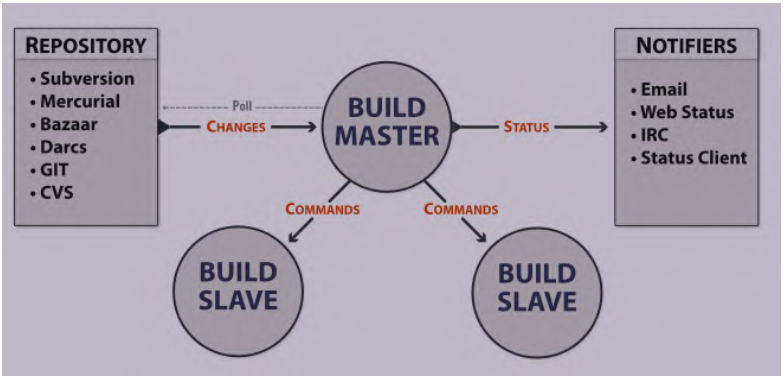
Remote File Inclusion merupakan sebuah serangan yang memaksa aplikasi web untuk mengakses dan mengeksekusi *file* dari tempat lain yang bisa saja berisi kode berbahaya. Penyerang memanfaatkan celah masukan aplikasi seperti URL atau formulir yang menyebabkan aplikasi mengikutsertakan suatu *file* berdasarkan masukan yang diberikan. Jika masukan tersebut diubah menjadi alamat *file* yang berisi kode berbahaya maka aplikasi akan mengeksekusi kode berbahaya tersebut.

2.9 PHP

PHP (akronim dari *PHP: Hypertext Preprocessor*) merupakan sebuah bahasa pemrograman *server-side* yang umum digunakan dalam pembuatan aplikasi berbasis web. PHP bisa dikombinasikan dengan HTML membentuk sebuah halaman web. Menurut hasil survey dari penyedia layanan survey seperti GitHut, RedMonk dan IEEE Spectrum, PHP masih menjadi bagian dari 10 besar bahasa pemrograman yang populer digunakan untuk pemrograman web.

2.10 Buildbot

Buildbot merupakan sebuah kerangka kerja berbasis *open-source* yang bisa digunakan untuk proses *Continuous Integration*. Kerangka kerja ini dibangun menggunakan bahasa pemrograman Python dan sudah dirancang untuk bisa dikembangkan sesuai kebutuhan. Berikut ini merupakan arsitektur dari kerangka kerja Buildbot.



Gambar 2.1: Arsitektur Kerangka Kerja Buildbot

2.11 W3af

Web Application Attack and Audit Framework (w3af) merupakan sebuah kerangka kerja yang digunakan untuk melakukan proses audit dan mengeksploitasi celah keamanan pada aplikasi berbasi web. W3af memiliki beberapa fitur unggulan yang menjadi pertimbangan penulis sehingga menggunakan kerangka kerja ini sebagai pendukung sistem yang akan dibuat. Fitur-fitur tersebut antara lain:

- Mudah untuk diintegrasikan dengan aplikasi lain.
- Mampu berjalan sebagai sebuah *web service*.
- Pilihan *output* hasil pemindaian yang beragam.
- Memiliki fitur *knowledge base* yang menyediakan informasi detail mengenai celah keamanan yang terdeteksi.

BAB 3

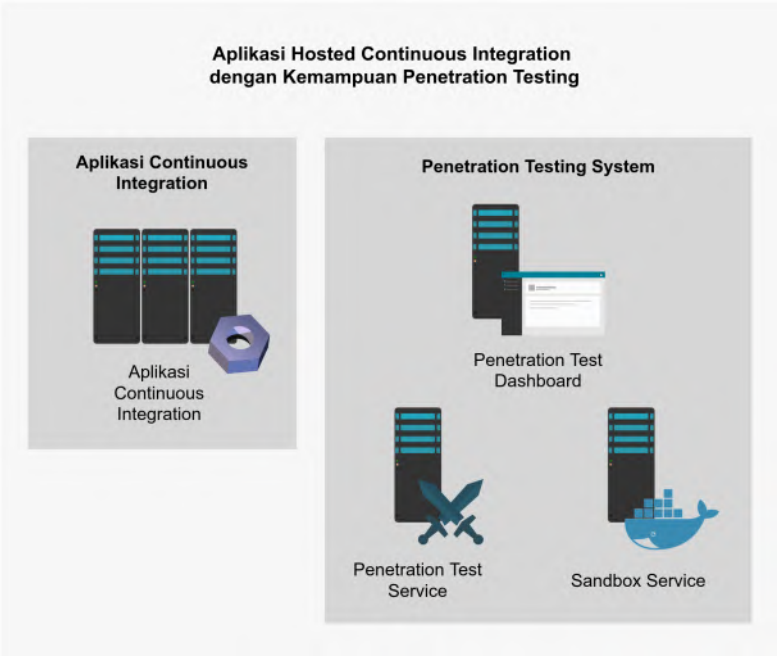
DESAIN DAN PERANCANGAN

Bab ini membahas mengenai analisis dan perancangan sistem, meliputi deskripsi umum sistem, diagram kasus penggunaan, fitur, proses utama, diagram alir, arsitektur, dan desain antarmuka.

3.1 Deskripsi Umum

Aplikasi *Hosted Continuous Integration* dengan kemampuan *penetration testing* merupakan sebuah sistem yang memudahkan pengembang aplikasi berbasis web dalam menjalankan serangkaian uji coba keamanan atau *penetration testing* terhadap aplikasi yang dibuat sebelum masuk tahap produksi. Pengujian yang dilakukan bisa berlangsung secara otomatis terjadwal maupun ketika sistem mendeteksi adanya perubahan kode sumber pada *Version Control System* (VCS). Sistem ini merupakan kolaborasi dari dua komponen, yakni aplikasi *Continuous Integration* dan *Penetration Testing System* sesuai Gambar 3.1. Aplikasi *Continuous Integration* berfungsi sebagai pusat kendali proses automasi, meliputi penentuan jadwal eksekusi proses *penetration testing*, pendeteksi perubahan kode sumber pada VCS serta pemicu komponen lain untuk bekerja. *Penetration Testing System* berfungsi sebagai sarana penyedia skenario serta media penyimpanan hasil *penetration testing*terhadap aplikasi. Sistem ini dibangun oleh tiga subsistem yang memiliki fungsi masing-masing, meliputi *Penetration Test Dashboard*, *Penetration Test Service* dan *Sandbox Service*. *Penetration Test Dashboard* berfungsi sebagai pusat kendali *Penetration Testing System* yang menyediakan antarmuka bagi pengguna untuk melakukan manajemen skenario dan hasil *penetration testing*. Subsistem inilah yang memicu subsistem lainnya untuk menjalankan fungsi masing-masing. *Penetration Test Service* berfungsi untuk menyediakan alat kakas *penetration testing*yang akan melakukan pemindaian celah keamanan

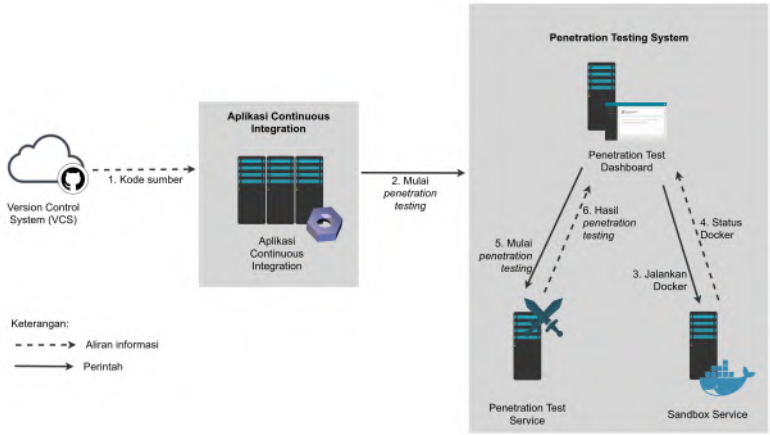
terhadap aplikasi target. *Sandbox Service* berfungsi sebagai sarana untuk menjalankan aplikasi web yang ingin diuji coba pada suatu *container* terisolasi. *Sandbox Service* menggunakan kemampaun Docker untuk mendukung fungsionalitasnya.



Gambar 3.1: Skema Komponen Sistem

Fungsionalitas sistem dimulai ketika aplikasi *Continuous Integration* mendeteksi adanya perubahan kode sumber pada VCS atau tercapainya jadwal untuk melakukan *penetration testing*. Aplikasi tersebut kemudian memicu *Penetration Testing System* untuk memulai aktivitas *penetration testing*. Ketika menerima perintah tersebut, subsistem *Penetration Test Dashboard* mengirimkan perintah kepada *Sandbox Service* untuk

menjalankan aplikasi web pada lingkungan Docker. Setelah aplikasi web berjalan dalam Docker, *Penetration Test Dashboard* kembali mengirim perintah kepada *Penetration Test Service* untuk memulai skenario *penetration testing* dengan cara melancarkan pola-pola serangan terhadap aplikasi web tersebut. Hasil uji coba diolah dan disimpan ke dalam basis data oleh *Penetration Test Service* untuk kemudian ditampilkan kepada pengguna atau aplikasi *Continuous Integration*. Data hasil uji coba inilah yang bisa digunakan pengembang dalam mengevaluasi dan memperbaiki kode program, sehingga tidak menimbulkan masalah di kemudian hari. Aliran informasi dan perintah yang terjadi dalam sistem tertera pada Gambar 3.2.

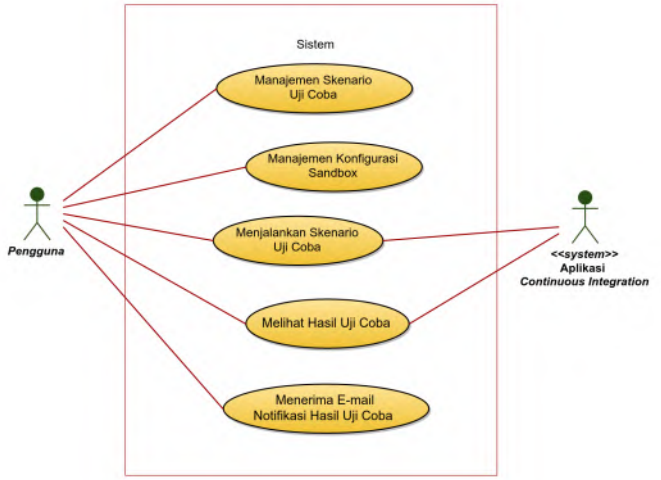


Gambar 3.2: Aliran Informasi dan Perintah pada Sistem

3.2 Diagram Kasus Penggunaan

Sistem yang dibangun berupa aplikasi web dengan fitur sesuai skenario kasus penggunaan yang tertera pada Gambar 3.3.

Terdapat dua aktor yang terlibat dengan sistem, yakni pengguna yang merupakan pengembang aplikasi dan memiliki kepentingan akan uji coba serta aplikasi *Continuous Integration* yang menggunakan kemampuan sistem ini untuk melakukan uji coba *penetration testing* terhadap suatu aplikasi.



Gambar 3.3: Digram Kasus Penggunaan

Diagram pada Gambar 3.3 menjelaskan kasus penggunaan pada sistem dan dideskripsikan masing-masing pada Tabel 3.1.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-001	Manajemen skenario uji coba	Daftar skenario uji coba disajikan dalam bentuk tabel beserta tombol aksi tambah, sunting, serta hapus. Pengguna bisa melakukan aksi tambah skenario, sunting skenario yang sudah ada serta hapus skenario.
UC-002	Manajemen konfigurasi *sandbox*	Pengguna bisa mengunggah *file* konfigurasi untuk aplikasi web yang ingin diuji coba, sehingga pada saat *sandbox* diaktifkan, *file-file* konfigurasi tersebut akan menggantikan *file* konfigurasi aplikasi yang dikloning dari VCS. Proses ini berfungsi untuk menyediakan *file* konfigurasi versi uji coba atau *dummy* selama aktivitas *penetration testing*.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-003	Menjalankan skenario uji coba	Terdapat sebuah rute yang bisa diakses menggunakan protokol HTTP untuk memicu berjalannya skenario uji coba. Jika aktor yang ingin menjalankan skenario adalah pengguna, dapat mengakses rute tersebut melalui sebuah tombol yang disediakan dalam antar muka sistem, sedangkan jika aktor yang menjalankan skenario adalah sistem lain maka sistem tersebut hanya perlu mengakses rute yang disediakan dengan metode POST.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-004	Melihat hasil uji coba	Hasil uji coba *penetration testing* dibuat dalam format JSON. Jika aktor yang melihat hasil tersebut adalah pengguna maka sistem menampilkannya dalam bentuk tabel, sedangkan jika aktor yang melihat hasil adalah sistem lain maka data tetap dalam bentuk JSON dan bisa diakses melalui sebuah rute khusus dengan metode GET.
UC-005	Menerima *e-mail* notifikasi hasil uji coba	Pengguna bisa menerima notifikasi hasil *penetration testing* melalui *e-mail* setelah proses uji coba selesai.

3.3 Fungsionalitas Sistem

Aplikasi yang dibangun pada Tugas Akhir ini memiliki fungsionalitas yang mendukung skenario kasus penggunaan. Fungsionalitas tersebut tersebar pada setiap subsistem yang meliputi:

a Fungsionalitas pada subsistem *Sandbox Service*:
- Kloning aplikasi dari VCS secara otomatis

Fungsi ini berfungsi untuk membuat salinan kode sumber aplikasi web dari VCS pada lingkungan uji coba.

- Eksekusi aplikasi pada lingkungan *sandbox* secara otomatis
  Fungsi ini berfungsi untuk menjalankan aplikasi web yang telah disalin dari VCS secara virtual pada lingkungan *server* produksi terisolasi, sehingga aplikasi bisa diakses seperti layaknya dalam lingkungan produksi. Lingkungan virtual tersebut didukung oleh aplikasi Docker.

b Fungsionalitas pada subsistem *Penetration Test Service*:

- Penetration testing
  Fungsi ini merupakan fitur utama yang berfungsi untuk melancarkan pola-pola serangan terhadap aplikasi web yang berjalan pada lingkungan *sandbox*. Serangan tidak akan mempengaruhi lingkungan lainnya sebab hanya difokuskan pada aplikasi web yang sudah terisolasi.

c Fungsionalitas pada subsistem *Penetration Test Dashboard*:

- Manajemen skenario uji coba
  Fungsi ini memungkinkan pengguna untuk membuat skenario uji coba yang meliputi aplikasi apa yang akan diuji, alamat repositori serta profil serangan yang dipilih. Melalui fitur ini pula pengguna bisa mengeksekusi perintah untuk memulai proses *penetration testing*.
- Laporan hasil uji coba
  Fungsi ini berfungsi untuk menangkap hasil *penetration testing* dari subsistem *Penetration Test Service* yang kemudian diolah dan ditampilkan kepada pengguna.

d Fungsi penjadwalan dan deteksi perubahan kode sumber
Fungsi ini berfungsi untuk menentukan jadwal eksekusi satu rangkaian uji coba terhadap aplikasi web berdasarkan jadwal pasti yang telah ditentukan maupun perubahan kode sumber yang terjadi pada VCS. Fungsi ini sudah disediakan oleh aplikasi *Continuous Integration* dan siap diintegrasikan dengan tiga subsistem di atas.

e Fungsi notifikasi hasil uji coba
Fungsi ini berfungsi untuk memberikan notifikasi melalui *e-mail* ketika suatu rangkaian uji coba selesai dilakukan. Fungsi ini sudah tersedia pula pada aplikasi *Continuous Integration*.

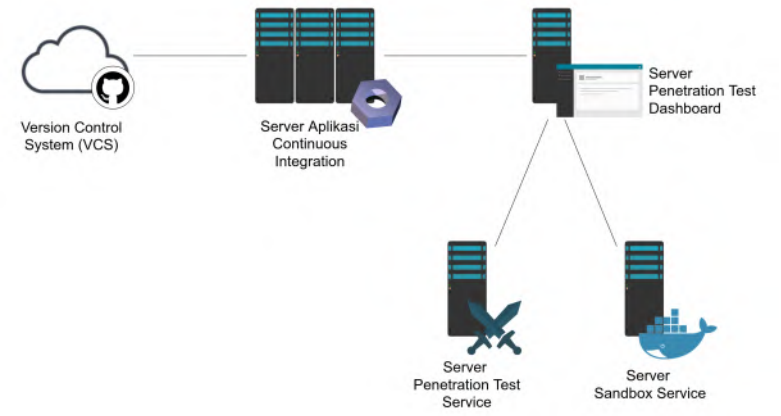
3.4 Arsitektur Sistem

Arsitektur sistem terdiri dari dua bagian, yakni arsitektur aplikasi serta arsitektur jaringan yang digunakan untuk menjalankan sistem.

3.4.1 Arsitektur Aplikasi

Hosted Continuous Integration yang akan dibangun terdiri dari tiga subsistem yang menjalankan fungsionalitas utama serta satu aplikasi *Continuous Integration* sesuai dengan Gambar 3.4. *Server Continuous Integration* berfungsi sebagai penerima informasi perubahan kode sumber aplikasi dari VCS, menentukan jadwal eksekusi skenario uji coba *penetration testing*, dan memerintahkan *Penetration Test Dashboard* untuk mengeksekusi skenario uji coba.

Server Penetration Test Dashboard berfungsi sebagai pusat pengaturan fungsionalitas sistem, meliputi pengaturan profil serangan, pengaturan skenario uji coba serta menjalankan skenario uji coba. Profil serangan menyediakan informasi mengenai langkah-langkah serangan yang akan dilancarkan pada



Gambar 3.4: Arsitektur Aplikasi

suatu skenario uji coba. Skenario uji coba menyediakan informasi mengenai nama aplikasi, alamat repositori VCS serta profil serangan yang akan digunakan. Ketika skenario uji coba dijalankan, *Penetration Test Dashboard* memerintahkan *Server Penetration Test Service* untuk melontarkan pola-pola serangan sesuai profil yang dipilih terhadap aplikasi web yang sudah dijalankan pada lingkungan *sandbox* pada *Server Sandbox Service*.

Server Penetration Test Service menyediakan *web service* untuk melancarkan serangan-serangan dalam skenario *penetration testing*. Terdapat beberapa aplikasi *penetration testing* yang berjalan pada *server* ini dan siap menerima perintah untuk menyerang aplikasi web yang berjalan pada *sandbox*.

Server Sandbox Service berfungsi sebagai lingkungan *sandbox* untuk menjalankan aplikasi web seolah-olah seperti berada pada lingkungan produksi. Pola-pola serangan akan dilancarkan pada aplikasi web yang berjalan pada lingkungan *sandbox* ini. *Sandbox* menyebabkan serangan tidak akan

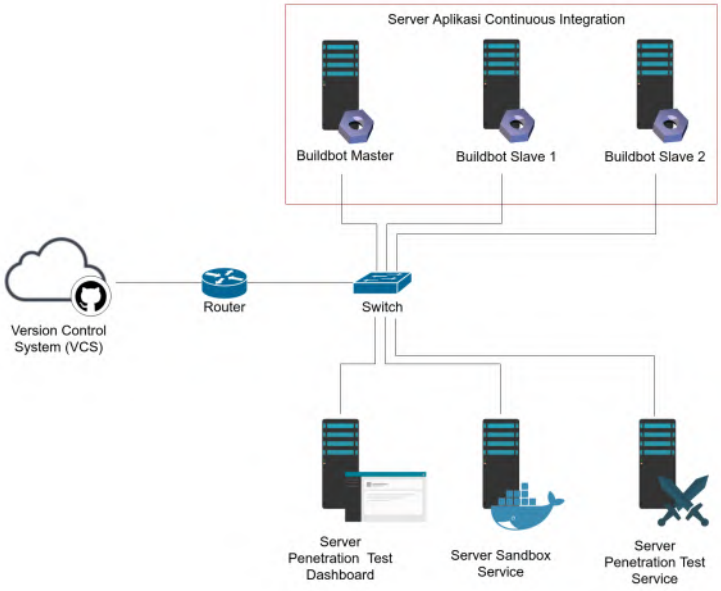
mempengaruhi lingkungan serta aplikasi lain karena aplikasi yang diserang berjalan pada lingkungan yang terisolasi. Server ini didukung oleh aplikasi Docker untuk menjalankan fungsionalitas *sandbox*.

3.4.2 Arsitektur Jaringan

Subsistem pada sistem yang dibuat pada Tugas Akhir ini berada pada satu jaringan yang sama. Terdapat enam buah server yang memiliki fungsi berbeda-beda seperti yang terlihat pada Gambar 3.5. Tiga buah server berfungsi sebagai aplikasi *Continuous Integration* yang didukung oleh kerangka kerja *Continuous Integration* Buildbot. Buildbot memiliki layanan Buildbot Master sebagai manajer segala fungsionalitasnya serta Buildbot Slave sebagai *worker* yang menjalankan segala macam uji coba. Sistem ini menjalankan satu buah Buildbot Master dan dua buah Buildbot Slave. Buildbot Master bertugas menerima perubahan kode sumber dari VCS dan memerintahkan Buildbot Slave untuk melakukan uji coba terhadap aplikasi web. Buildbot Slave 1 berfungsi sebagai *worker* untuk menjalankan *unit test* dan *functional test*. *Unit test* dan *functional test* tidak akan dibahas terlalu detail karena merupakan rangkaian uji coba kesesuaian aplikasi terhadap daftar kebutuhan. Buildbot Slave 2 berfungsi sebagai *worker* untuk menjalankan uji coba *penetration testing*. Pada *worker* kedua inilah ditanamkan *script* yang memicu subsistem *Penetration Test Dashboard* untuk memulai *penetration testing*.

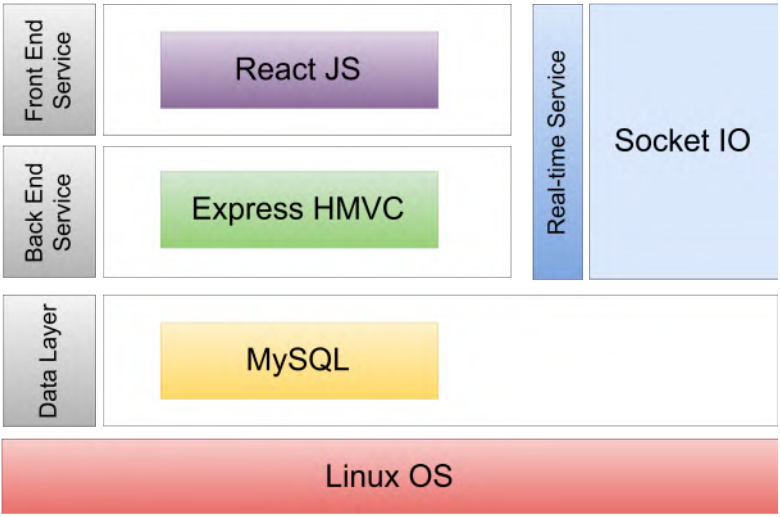
3.4.3 Desain *Server Penetration Test Dashboard*

Penetration Test Dashboard merupakan sebuah layanan berbasis web yang menyediakan antar muka bagi *developer* untuk melakukan manajemen terkait proses uji coba keamanan aplikasi. Subsistem ini dibangun menggunakan kerangka kerja



Gambar 3.5: Desain Arsitektur Jaringan

Express JS, React JS, basis data MySQL serta pustaka Socket.IO untuk mendukung protokol komunikasi websocket. Protokol websocket digunakan untuk mengirim hasil uji coba secara *real-time* ketika proses *penetration testing* sedang berlangsung. Diagram arsitektur *Penetration Test Dashboard* tertera pada Gambar 3.6.

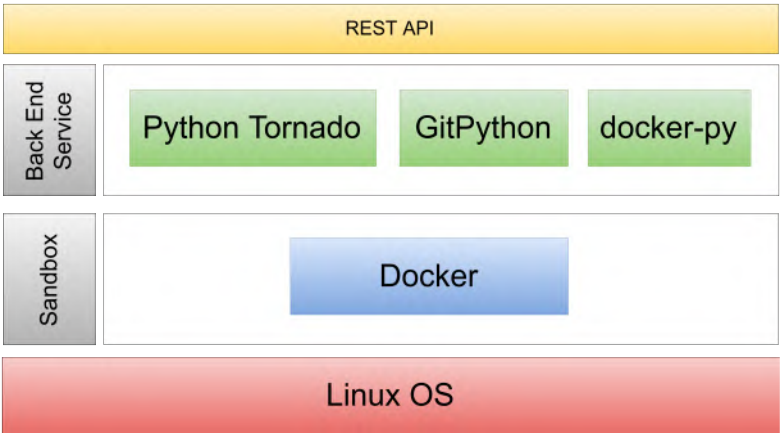


Gambar 3.6: Diagram Arsitektur *Penetration Test Dashboard*

3.4.4 Desain *Server Sandbox Service*

Sandbox Service merupakan sebuah *server* yang berfungsi sebagai penyedia lingkungan virtual atau *sandbox* untuk menjalankan aplikasi web yang ingin diuji coba. *Sandbox* memungkinkan aplikasi web berjalan seperti layaknya pada lingkungan produksi tanpa mempengaruhi aplikasi lain yang berjalan pada sistem operasi yang sama. *Sandbox Service* dibangun menggunakan Python Tornado sebagai *web service*

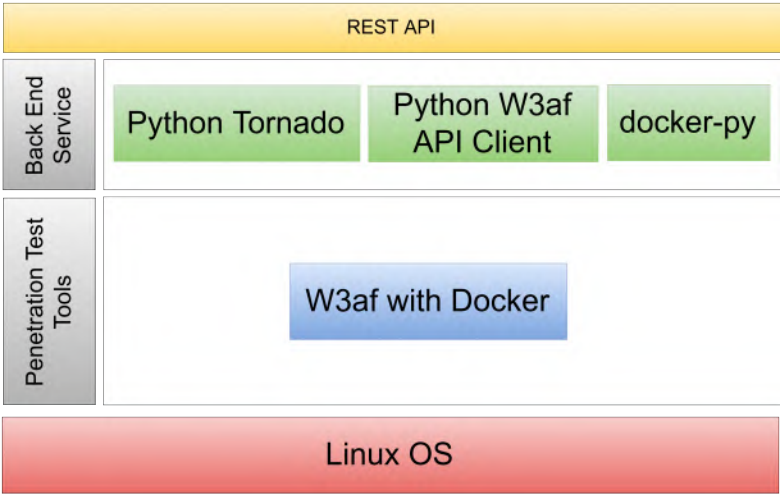
yang menerima perintah aktivasi *sandbox* dari subsistem *Penetration Test Dashboard* dan didukung oleh pustaka GitPython untuk melakukan kloning kode sumber dari VCS serta pustaka docker-py untuk berinteraksi dengan Docker. Aplikasi Docker digunakan sebagai sarana untuk menjalankan *sandbox* bagi tiap aplikasi web yang ingin diuji coba. Diagram arsitektur *Sandbox Service* tertera pada Gambar 3.7.



Gambar 3.7: Diagram Arsitektur *Sandbox Service*

3.4.5 Desain *Server Penetration Test Service*

Fungsi utama dari sistem terdapat pada subsistem *Penetration Test Service*. Subsistem ini menyediakan aplikasi *penetration testing* yang akan melancarkan serangan-serangan pada aplikasi web yang akan diuji coba. Aplikasi *penetration testing* yang berjalan pada subsistem ini adalah W3af. W3af berjalan sebagai *web service* dan bisa menerima perintah dari *Penetration Test Dashboard*. Diagram arsitektur *Penetration Test Service* tertera pada Gambar 3.8.



Gambar 3.8: Diagram Arsitektur *Penetration Test Service*

3.5 Diagram Alir

Subbab ini membahas mengenai diagram alir yang dibagi menjadi tiga bagian, meliputi *Penetration Test Dashboard*, *Sandbox Service* dan *Penetration Test Service*.

3.5.1 Diagram Alir *Penetration Test Dashboard*

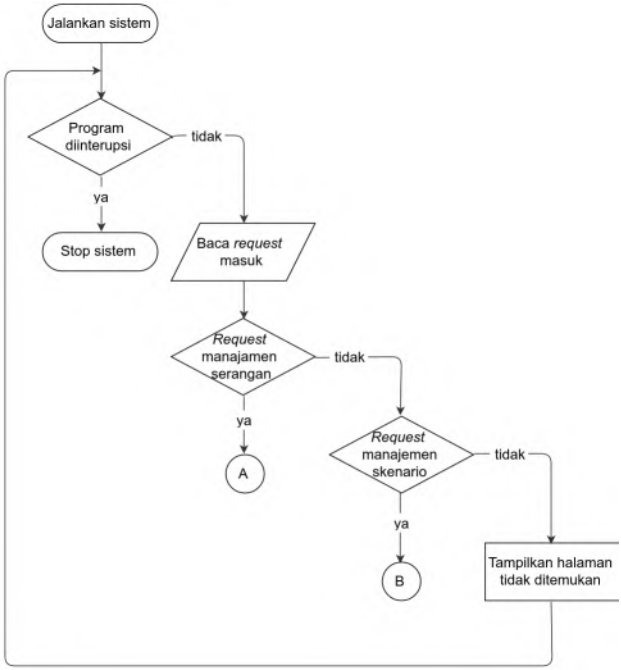
Penetration Test Dashboard merupakan server pusat kendali segala aktivitas uji coba keamanan. Sistem ini berjalan sebagai sebuah *web service* dan memiliki antar muka pengguna berupa tampilan dasbor. Secara umum cara kerjanya adalah menunggu datangnya *request* lalu mencarikan halaman yang sesuai. Terdapat dua fitur utama yang disediakan *Penetration Test*

Dashboard, meliputi manajemen serangan dan manajemen skenario. Diagram alir sistem ini secara umum tertera pada Gambar 3.9, kemudian diperjelas pada Gambar 3.10 untuk fitur manajemen serangan serta Gambar 3.11 untuk fitur manajemen skenario.

Manajemen serangan menampilkan daftar profil serangan dalam bentuk tabel beserta aksi untuk *create*, *update* dan *delete*. Ketika fitur *create* dijalankan, sistem menampilkan formulir isian profil serangan baru sedangka untuk aksi *update*, sistem menampilkan formulir berisi informasi yang sudah tersimpan pada basis data. Aksi *delete* menyebabkan data tertentu dihapus dari basis data. Hal yang sama juga berlaku untuk fitur manajemen skenario. Namun, pada manajemen skenario terdapat aksi jalankan skenario untuk memerintahkan *Sandbox Service* dan *Penetration Test Service* bekerja.

3.5.2 Diagram Alir *Sandbox Service*

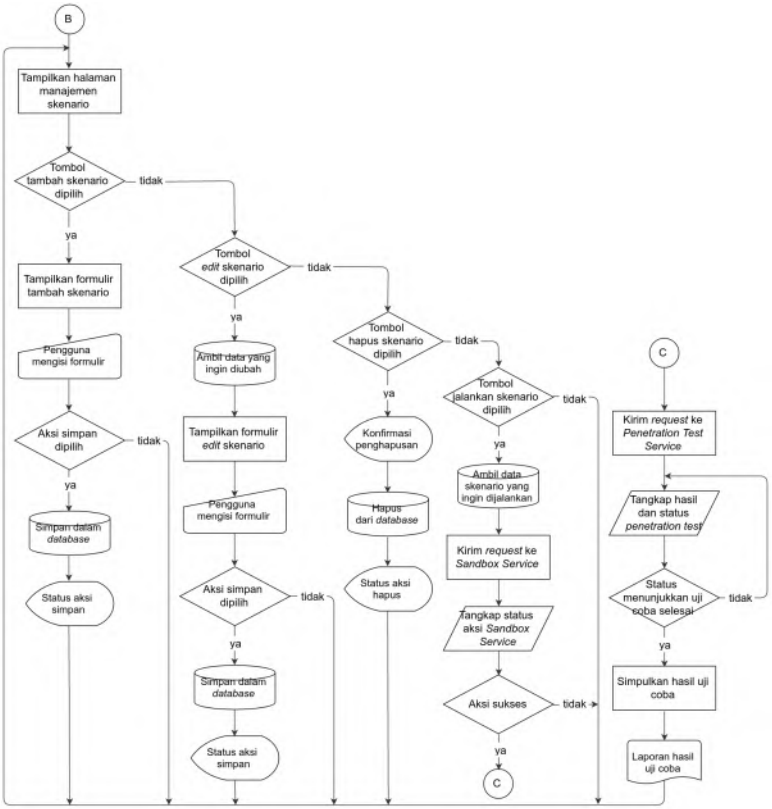
Sandbox Service berjalan sebagai sebuah *web service* yang siap menerima perintah dari *Penetration Test Dashboard*. Sistem akan membaca *request* yang masuk kemudian memastikan *request paramater* memenuhi ketentuan. Suatu *request* memenuhi ketentuan jika berisi informasi nama aplikasi serta alamat repositori VCS. Jika *request parameter* tidak memenuhi ketentuan, maka *requst* tidak dilayani. Namun, jika *request parameter* memenuhi ketentuan, maka sistem akan memeriksa daftar *running task* untuk memastikan apakah *request* sedang dalam proses eksekusi. Satu *request* hanya bisa dieksekusi sekali dalam satu waktu untuk menghindari konflik saat pembentukan *sandbox*. Untuk menjalankan *sandbox*, sistem harus memastikan bahwa kode sumber aplikasi sudah diunduh dari VCS ke direktori lokal. Sistem secara otomatis menentukan apakah aksi yang dilakukan adalah *clone* atau *pull*. Jika aplikasi belum tersedia pada direktori lokal maka sistem akan menjalankan



Gambar 3.9: Diagram Alir *Penetration Test Dashboard* Secara Umum

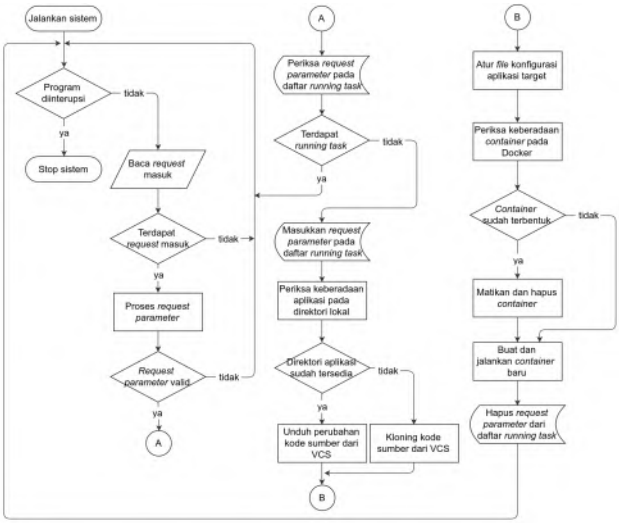
perintah *clone*, sedangkan jika aplikasi sudah tersedia maka sistem cukup menjalankan perintah *pull* untuk mendapatkan kode sumber terbaru. Selanjutnya sistem akan mengatur *file* konfigurasi aplikasi berdasarkan opsi yang sudah ditentukan oleh pengguna. *File* konfigurasi yang dimaksud adalah beberapa *file* yang harus dimodifikasi setelah aplikasi diunduh dari VCS agar bisa berjalan dengan baik, misalnya *file* konfigurasi basis data. Setelah kode sumber siap, sistem akan menjalankan *container* untuk aplikasi menggunakan Docker. Jika sebelumnya sudah terdapat *container* untuk aplikasi yang akan dijalankan, maka *container* tersebut dinonaktifkan lalu dihapus dan dibuat kembali demi memastikan kesesuaiannya dengan kode sumber terbaru.

Gambar 3.10: Diagram Alir Manajemen Serangan pada *Penetration Test Dashboard*



Gambar 3.11: Diagram Alir Manajemen Skenario pada *Penetration Test Dashboard*

Sistem secara otomatis menentukan alamat berbeda untuk setiap *container* yang berjalan dengan cara membedakan *port* layanan, sementara alamat IP tetap menggunakan IP *Sandbox Service*. Pada tahap terakhir, sistem menghapus *request* dari daftar *running task* yang menandakan bahwa *request* telah selesai dieksekusi. Gambar 3.12 menunjukkan diagram alir dari *Sandbox Service*.



Gambar 3.12: Diagram Alir *Sandbox Service*

3.5.3 Diagram Alir *Penetration Test Service*

Penetration Test Service berjalan sebagai web service yang siaga untuk menerima perintah dari *Penetration Test Dashboard*. Ketika *request* diterima, sistem akan memeriksa apakah *request* berupa perintah untuk melakukan *penetration testing* atau permintaan hasil *penetration testing*. Setiap *request* harus menyertakan paramater yang menunjukkan alamat target

penetrasi. Alamat target disediakan oleh *Penetration Test Dashboard* dan merupakan alamat dari *sandbox* aplikasi yang sedang berjalan. Jika *request* berupa perintah untuk *penetration testing* maka sistem akan melancarkan serangan-serangan terhadap alamat target. Sedangkan, jika *request* berupa permintaan hasil, maka sistem akan memberikan hasil *penetration testing* terhadap alamat target dalam format JSON. Gambar 3.13 menunjukkan diagram alir untuk *Penetration Test Service*.

3.6 Rancangan Antarmuka

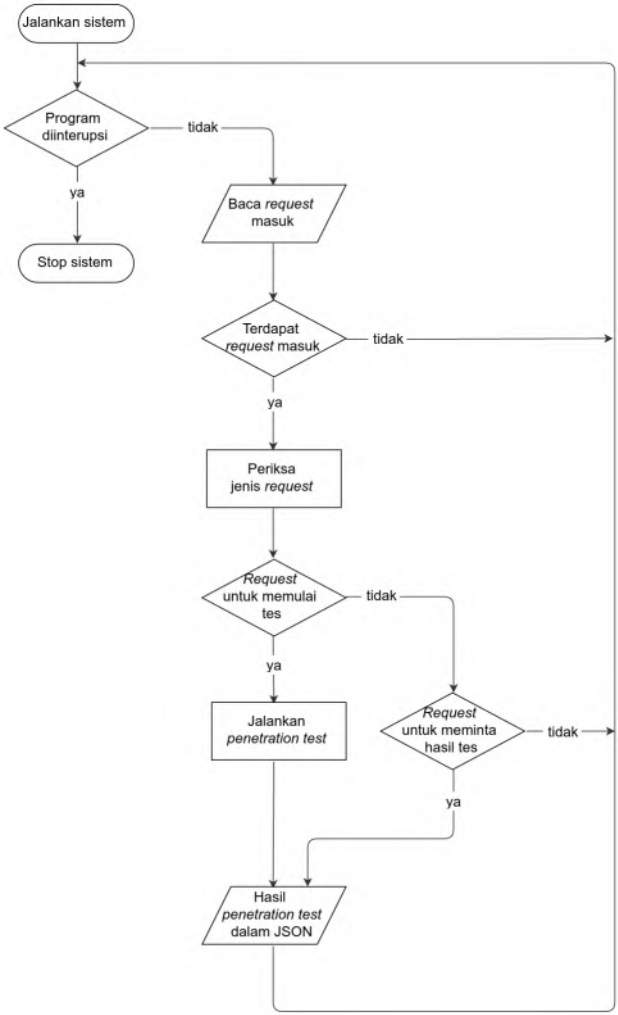
Subbab ini membahas mengenai rancangan antarmuka sistem, meliputi rancangan antarmuka *Penetration Test Dashboard*, *Sandbox Service*, *Penetration Test Service*.

3.6.1 Antarmuka *Penetration Test Dashboard*

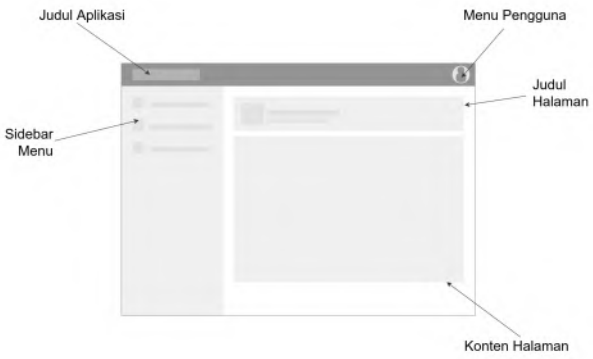
Penetration Test Dashboard memiliki antarmuka pengguna berupa halaman dasbor yang dibangun menggunakan kerangka kerja React JS. Terdapat templat dasbor yang dipakai pada semua halaman yang disediakan sistem. Pada templat tersebut tersedia bagian judul dan konten halaman yang dinamis dan bisa diisi sesuai kebutuhan. Dua bagian inilah yang nantinya akan berubah-ubah sesuai halaman yang dibuka pengguna. Desain antarmuka templat dasbor tertera pada Gambar 3.14.

Fitur manajemen serangan menampilkan sebuah tabel berisi daftar profil serangan beserta beberapa tombol aksi. Berikut ini merupakan daftar komponen yang terdapat pada antarmuka manajemen serangan beserta fungsinya:

- Tombol tambah profil serangan berfungsi untuk menampilkan formulir tambah profil serangan baru.
- Tombol hapus profil serangan berfungsi untuk menghapus data pada baris yang ditandai.



Gambar 3.13: Diagram Alir *Penetration Test Service*



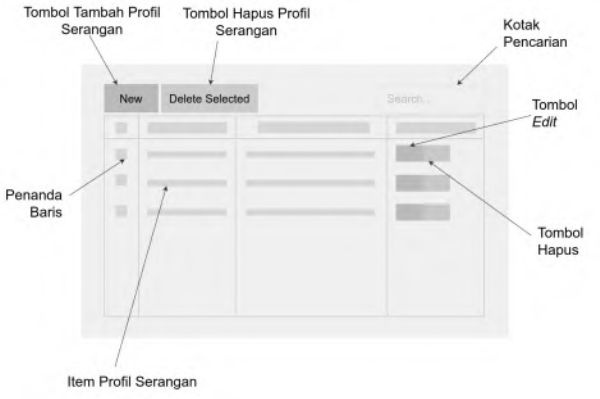
Gambar 3.14: Desain Antarmuka Templat Dasbor pada *Penetration Test Dashboard*

- Kotak pencarian berfungsi sebagai sarana mencari data dengan kata kunci tertentu.
- Tombol *edit* pada setiap baris berfungsi untuk menampilkan formulir *edit* data untuk baris tersebut.
- Tombol hapus pada setiap baris berfungsi untuk menghapus data pada baris tersebut.
- Item profil serangan merupakan baris yang menunjukkan detail dari data yang ada pada basis data.
- Item penanda baris berfungsi untuk menandai baris yang akan dihapus. Penanda baris ini berkolaborasi dengan tombol hapus profil untuk menjalankan fungsinya.

Desain antarmuka manajemen serangan tertera pada Gambar 3.15.

Fitur manajemen skenario menampilkan sebuah tabel berisi daftar skenario serangan beserta beberapa tombol aksi. Berikut ini merupakan daftar komponen yang terdapat pada antarmuka manajemen skenario beserta fungsinya:

- Tombol tambah skenario berfungsi untuk menampilkan formulir tambah skenario serangan baru.
- Tombol hapus skenario berfungsi untuk menghapus data



Gambar 3.15: Desain Antarmuka Manajemen Serangan pada *Penetration Test Dashboard*

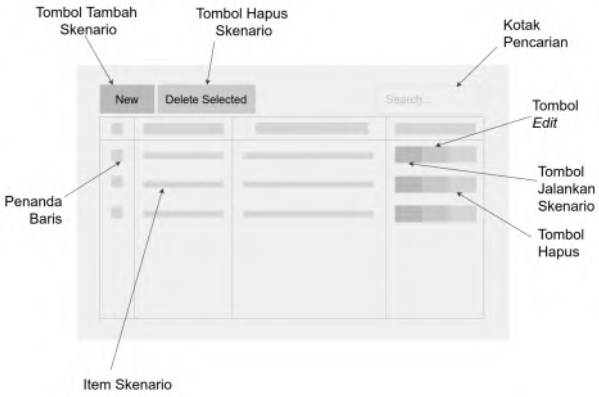
pada baris yang ditandai.

- Kotak pencarian berfungsi sebagai sarana mencari data dengan kata kunci tertentu.
- Tombol *edit* pada setiap baris berfungsi untuk menampilkan formulir *edit* data untuk baris tersebut.
- Tombol hapus pada setiap baris berfungsi untuk menghapus data pada baris tersebut.
- Item skenario merupakan baris yang menunjukkan detail dari data yang ada pada basis data.
- Item penanda baris berfungsi untuk menandai baris yang akan dihapus. Penanda baris ini berkolaborasi dengan tombol hapus skenario untuk menjalankan fungsinya.

Desain antarmuka manajemen skenario tertera pada Gambar 3.16.

3.6.2 Antarmuka *Sandbox Service*

Sandbox Service tidak memiliki antarmuka grafis karena tidak langsung berinteraksi dengan pengguna, melainkan berinteraksi dengan *Penetration Test Dashboard*. Antarmuka



Gambar 3.16: Desain Antarmuka Manajemen Skenario pada *Penetration Test Dashboard*

yang bisa diakses oleh *Penetration Test Dashboard* berupa rute *web service* yang harus diakses menggunakan protokol HTTP. Rute yang disediakan adalah sebuah alamat yang harus diakses menggunakan metode POST disertai parameter nama aplikasi serta alamat repositori VCS. Jika rute ini diakses menggunakan metode sesuai ketentuan, maka sistem akan menjalankan aksinya untuk mengaktifkan *sandbox* bagi aplikasi target, kemudian mengirimkan status aksi dalam format JSON.

3.6.3 Antarmuka *Penetration Test Service*

Penetration Test Service memiliki karakteristik serupa dengan *Sandbox Service* yakni berupa *web serivce* tanpa ada antarmuka grafis untuk pengguna. Rute yang disediakan lebih beragam karena terdapat rute untuk menjalankan aksi *penetration testing* dan rute untuk mendapatkan hasil dari *penetration testing*.

(Halaman ini sengaja dikosongkan)

BAB 4

IMPLEMENTASI

Bab ini membahas implementasi sistem *Hosted Continuous Integration* secara rinci, meliputi lingkungan implementasi serta rincian implementasi tiap subsistem.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dibagi menjadi dua bagian, meliputi perangkat lunak dan perangkat keras.

4.1.1 Perangkat Lunak

Lingkungan implementasi dan pengembangan dilakukan menggunakan perangkat lunak sebagai berikut :

- Sistem Operasi Linux Ubuntu Server 14.04.04 LTS sebagai lingkungan implementasi *Sanbox Service*, *Penetration Test Service* dan Buildbot.
- Sistem Operasi Linux Ubuntu Desktop 14.04.04 LTS sebagai lingkungan implementasi *Penetration Test Dashboard* dan pengembangan sistem secara keseluruhan.
- Editor teks Vim untuk pengembangan *script* pada *server* dan Sublime Text 2 untuk pengembangan sistem secara keseluruhan.
- Git versi 1.9.1 untuk manajemen versi aplikasi selama pengembangan serta mendukung kemampuan *Sandbox Service*.
- NodeJS versi 6.2.1 untuk mendukung berjalannya *Penetration Test Dashboard*.
- NPM versi 3.9.3 untuk manajemen paket NodeJS yang dibutuhkan pada *Penetration Test Dashboard*.
- Bower versi 1.7.7 untuk manajemen aset yang dibutuhkan pada antar muka grafis *Penetration Test Dashboard*.
- Nodemon versi 1.9.2 untuk menjalankan NodeJS selama proses pengembangan *Penetration Test Dashboard*.

- Python versi 2.7.6 untuk mendukung berjalannya *Sandbox Service* dan Buildbot.
- Python Virtualenv versi 15.0.1 untuk mendukung berjalannya Buildbot.
- MySQL versi 14.14 distribusi 5.5.49 untuk basis data pada *Penetration Test Dashboard* serta *Sandbox Service*.
- Buildbot versi 0.9.0b8 sebagai aplikasi *Continuous Integration*.
- Python Tornado versi 4.3 untuk menjalankan *Sandbox Service*.
- Docker versi 1.11.1 untuk menjalankan lingkungan virtual pada *Sandbox Service*.
- Express JS versi 4.13.1 untuk web server *Penetration Test Dashboard*.
- React JS versi 15.1.0 untuk antar muka grafis pada *Penetration Test Dashboard*.
- W3af sebagai alat kakas untuk melancarkan *penetration testing*.
- Paket LaTex untuk pembuatan buku tugas akhir.
- Peramban *web* Mozilla Firefox dan Google Chrome untuk uji coba sistem.

Selain itu, pengerjaan Tugas Akhir ini juga menggunakan pustaka berikut:

- GitPython versi 2.0.3 untuk mendukung interaksi *Sandbox Service* dengan Git.
- Docker-py versi 1.8.1 untuk mendukung interaksi *Sandbox Service* dengan Docker.
- Bootstrap versi 3.3.6 untuk mendukung antar muka grafis pada *Penetration Test Dashboard*.
- React Bootstrap Table versi 2.0.7 untuk mendukung antar muka berupa daftar item pada *Penetration Test Dashboard*.
- Socket.IO versi 1.4.6 untuk mendukung kemampuan *real-time* pada saat pembacaan hasil *penetration testing*

oleh *Penetration Test Dashboard*.

4.1.2 Perangkat Keras

Lingkungan perangkat keras yang digunakan selama proses pengerjaan Tugas Akhir ini adalah sebagai berikut:

- Komputer Aspire M3970 dengan *processor* empat inti Intel(R) Core(TM) i3-2120, 4GB RAM dan sistem operasi Linux Ubuntu Desktop 14.04.04 LTS untuk menjalankan *Penetration Test Dashboard* dan mendukung proses pengembangan sistem secara keseluruhan.
- Server dengan *processor* empat inti Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz, 8GB RAM dan sistem operasi Proxmox untuk menjalankan beberapa subsistem pada *virtual machine*, meliputi:
  - *Sandbox Service* berjalan pada *virtual machine* dengan *processor* satu inti, 512MB RAM, dan sistem operasi Linux Ubuntu 14.04.4 LTS.
  - *Penetration Test Service* berjalan pada *virtual machine* dengan *processor* dua inti, 1GB RAM, dan sistem operasi Ubuntu 14.04.4 LTS.
  - Buildbot Master, Buildbot Slave 1 dan Buildbot Slave 2 masing-masing berjalan pada *virtual machine* terpisah dengan spesifikasi *processor* satu inti, 512MB RAM, dan sistem operasi Linux Ubuntu 14.04.4 LTS.

4.2 Rincian Implementasi *Penetration Test Dashboard*

Penetration Test Dashboard merupakan pusat kendali dari semua aktivitas *penetration testing*. Implementasi subsistem ini dibagi menjadi beberapa bagian, meliputi persiapan lingkungan implementasi, instalasi paket, pseudocode, dan antar muka.

4.2.1 Persiapan Lingkungan Implementasi

Persiapan lingkungan implementasi meliputi langkah-langkah sebagai berikut:

1. Instalasi komputer dengan sistem operasi Linux Ubuntu 14.04.4 LTS, teks editor VIM dan Sublime Text 2.
2. Konfigurasi *interface* jaringan dengan menggunakan alamat IP 10.151.36.30.
3. Instalasi Node JS, Node Packet Manager (NPM), Bower dan Nodemon.
4. Instalasi Git.
5. Instalasi MySQL sebagai basis data *Penetration Test Dashboard*.

4.2.2 Instalasi Paket

Instalasi paket meliputi langkah-langkah sebagai berikut:

1. Instalasi paket Express JS, Socket.IO JS serta paket-paket pendukung lainnya. Paket yang dibutuhkan oleh *Penetration Test Dashboard* diunduh menggunakan bantuan NPM dan terdokumentasi pada sebuah *file* dengan nama `package.json`. Proses mengunduh dan menginstalasi paket dilakukan dengan cara menjalankan perintah `npm install` pada direktori yang sama dengan keberadaan `package.json`, sehingga NPM bisa mengunduh paket sesuai kebutuhan yang tertera pada *file* tersebut. Isi dari `package.json` tertera pada Kode Sumber 4.1.

```
{
  "name": "Penetration Test Dashboard",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
```

```
  "dependencies": {
    "body-parser": "~1.13.2",
    "bower": "^1.7.9",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "express": "~4.13.1",
    "handlebars-form-helpers": "~0.1.3",
    "hbs": "^3.1.1",
    "morgan": "~1.6.1",
    "mysql": "^2.10.2",
    "ping": "^0.1.10",
    "request-json": "^0.5.6",
    "sequelize": "^3.19.1",
    "serve-favicon": "~2.3.0",
    "socket.io": "^1.4.6"
  }
}

```

Kode Sumber 4.1: Isi package.json pada *Penetration Test Dashboard*

2. Instalasi pustaka yang dibutuhkan oleh antar muka *Penetration Test Dashboard* dengan bantuan Bower dan terdokumentasi pada *file* `bower.json`. Proses ini dilakukan dengan cara menjalankan perintah `bower install` pada direktori yang sama dengan keberadaan `bower.json`, sehingga Bower akan mengunduh semua pustaka yang dibutuhkan sesuai isi dari *file* tersebut. Isi dari `bower.json` tertera pada Kode Sumber 4.2.

```
{
  "name": "public",
  "homepage": "https://github.com/sdarmaputra/
    pentest-dashboard",
  "authors": [
    "sdarmaputra <surya.igede@gmail.com>"
  ],
  "description": "Application assets",
  "main": "",
  "moduleType": [],
  "license": "MIT",
```

```
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
    "bootstrap": "^3.3.6",
    "react": "^15.1.0",
    "react-bootstrap-table": "^2.0.7",
    "babel": "*",
    "requirejs": "^2.2.0",
    "react-bootstrap": "^0.29.4"
  }
}

```

Kode Sumber 4.2: Isi bower.json pada *Penetration Test Dashboard*

4.2.3 Antar muka

Antar muka pengguna *Penetration Test Dashboard* sesuai dengan rancangan pada subbab 3.6. Belum tersedia.

4.3 Rincian Implementasi *Sandbox Service*

Sandbox Service merupakan *server* untuk menjalankan aplikasi target dalam lingkungan virtual terisolasi. Implementasi subsistem ini dibagi menjadi beberapa bagian, meliputi persiapan lingkungan implementasi, instalasi paket, pseudocode, dan rute *web service*.

4.3.1 Persiapan Lingkungan Implementasi

Persiapan lingkungan implementasi meliputi langkah-langkah berikut:

1. Pembuatan *virtual machine* dengan spesifikasi RAM 512MB dan satu inti *processor*.
2. Instalasi *virtual machine* dengan sistem operasi Linux Ubuntu 14.04.4 LTS dan editor VIM.
3. Konfigurasi *interface* jaringan dengan menggunakan alamat IP 10.151.36.93.
4. Instalasi Python, python-dev dan python-pip.
5. Instalasi Git sebagai sarana untuk melakukan kloning kode sumber aplikasi target dari VCS.
6. Instalasi Docker sebagai sarana untuk menjalankan aplikasi target pada linkungan virtual.
7. Instalasi MySQL sebagai basis data yang bisa digunakan oleh aplikasi target.

4.3.2 Instalasi Paket

Sandbox Service memerlukan paket-paket khusus yang menunjang fungsinya. Instalasi paket untuk *Sandbox Service* meliputi langkah-langkah berikut:

1. Instalasi Python Tornado sebagai *web server* untuk melayani permintaan aktivasi *sandbox*. Untuk menjalankan instalasi tersebut, perintah yang digunakan adalah `pip install tornado`.
2. Instalasi GitPython sebagai perantara Tornado dan Git dengan cara menjalankan perintah `pip install gitpython`.
3. Instalasi docker-py sebagai perantara Tornado dan Docker dengan cara menjalankan perintah `pip install docker-py`.

4.3.3 Rute *Web Service*

Sandbox Service tidak memiliki antar muka grafis karena tidak berinteraksi langsung dengan pengguna. Namun, subsistem

ini berinteraksi dengan *Penetration Test Dashboard*, sehingga diperlukan adanya rute-rute yang bisa diakses untuk memicu *Sandbox Service* menjalankan fungsinya. Daftar rute yang disediakan oleh *Sandbox Service* tertera pada Tabel 4.1.

Tabel 4.1: Daftar Rute *Sandbox Service*

HTTP Method	**Rute**	**Parameter**	**Deskripsi**
POST	/deploy	app_name, repo_name	Berfungsi untuk memicu aksi pembuatan *sandbox* untuk aplikasi dengan nama yang tertera pada paramter `app_name` dan mengambil kode sumber dari alamat yang tertera pada parameter `repo_name`

4.4 Rincian Implementasi *Penetration Test Service*

Penetration Test Service merupakan *server* untuk menjalankan *penetration testing* dengan cara melancarkan serangan-serangan ke aplikasi target dalam lingkungan virtual terisolasi. Implementasi subsistem ini dibagi menjadi beberapa bagian, meliputi persiapan lingkungan implementasi, instalasi alat kakas, pseudocode, dan rute *web service*.

4.4.1 Persiapan Lingkungan Implementasi

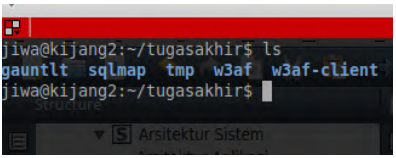
Persiapan lingkungan implementasi meliputi langkah-langkah berikut:

1. Pembuatan *virtual machine* dengan spesifikasi RAM 1GB dan dua inti *processor*.
2. Instalasi *virtual machine* dengan sistem operasi Linux Ubuntu 14.04.4 LTS dan editor VIM.
3. Konfigurasi *interface* jaringan dengan menggunakan alamat IP 10.151.36.92.
4. Instalasi Python, python-dev dan python-pip.
5. Instalasi Git sebagai sarana untuk melakukan kloning alat kakas dari penyedia.

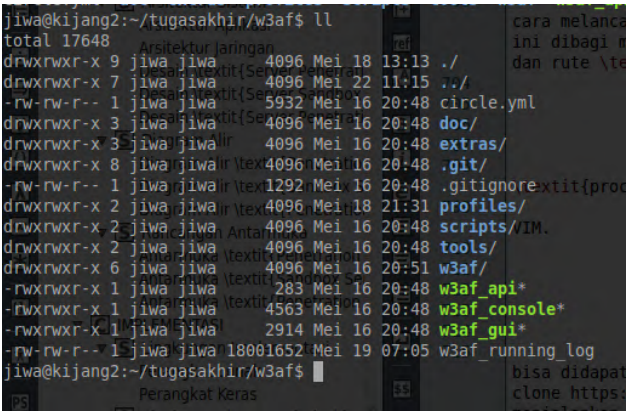
4.4.2 Instalasi Alat Kakas

Alat kakas yang digunakan sebagai alat untuk *penetration testing* adalah W3af. Aplikasi ini bisa didapatkan dengan cara melakukan kloning kode sumber dari repositori penyedia melalui perintah `git clone https://github.com/andresriancho/w3af.git`. Proses kloning menghasilkan direktori w3af sesuai Gambar 4.1. Setelah kloning berhasil, W3af sudah bisa digunakan dengan cara menjalankan *executable file* pada direktori aplikasinya. Isi direktori aplikasi W3af tertera pada Gambar 4.2. *Penetration Test Service* menggunakan aplikasi yang berjalan sebagai *web service* dan W3af menyediakan kemampuan tersebut. Untuk menjalankan W3af sebagai sebuah *web service*, *executable file* yang harus dijalankan adalah w3af_api dengan perintah `./w3af_api <alamat IP>:<port> &`. Alamat IP yang digunakan adalah alamat dari *server* menjalankan W3af, diisi dengan `0.0.0.0` jika W3af ingin diakses melalui alamat lokal maupun IP *server*. Setelah perintah tersebut dijalankan, maka W3af akan berjalan di balik layar sebagai sebuah *web service*

yang bisa diakses melalui alamat IP *server* dan port yang telah ditentukan seperti yang terlihat pada Gambar 4.3.



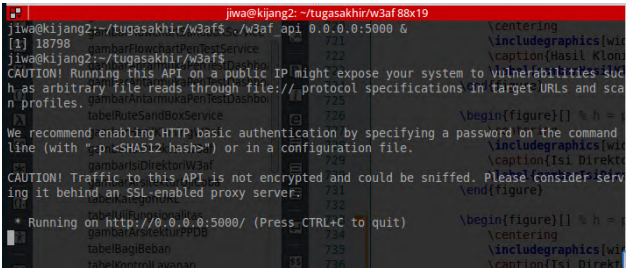
Gambar 4.1: Hasil Kloning W3af dari Repositori Berupa Direktori w3af



Gambar 4.2: Isi Direktori W3af

4.4.3 Rute *Web Service*

Penetration Test Service tidak memiliki antar muka grafis karena tidak berinteraksi langsung dengan pengguna. Namun, subsistem ini berinteraksi dengan *Penetration Test Dashboard*, sehingga diperlukan adanya rute-rute yang bisa diakses untuk memicu *Penetration Test Service* menjalankan fungsinya. Rute-rute yang disediakan oleh *Penetration Test Service* mengadopsi rute-rute yang disediakan oleh *web service* W3af.



Gambar 4.3: W3af Berjalan Sebagai *Web Service*

Daftar rute yang disediakan oleh *Penetration Test Service* tertera pada Tabel 4.2.

Tabel 4.2: Daftar Rute *Penetration Test Service*

HTTP Method	**Rute**	**Parameter**	**Deskripsi**
POST	/scans/	scan_profile, target_urls	Berfungsi untuk menjalankan proses *penetration testing* oleh W3af. `scan_profile` merupakan *file* profil serangan yang akan dijalankan. *File* ini sudah disediakan oleh W3af dan berisi daftar skenario serangan yang akan dilancarkan kepada target. `target_urls` berisi alamat aplikasi target serangan yang diisi dengan alamat *sandbox* dari aplikasi target.
GET	/scans/	-	Berfungsi untuk mendapatkan daftar proses *penetration testing* yang sedang berjalan.

Tabel 4.2: Daftar Rute *Penetration Test Service*

HTTP Method	**Rute**	**Parameter**	**Deskripsi**
GET	/scans/<scan-id>/kb/	-	Berfungsi untuk melihat hasil dari *penetration testing* tertentu berdasarkan pada `scan-id`.
GET	/scans/<scan-id>/status	-	Berfungsi untuk melihat status dari proses *penetration testing* tertentu berdasarkan pada `scan-id`.
GET	/scans/<scan-id>/stop	-	Berfungsi untuk menghentikan proses *penetration testing* tertentu berdasarkan pada `scan-id`.
DELETE	/scans/<scan-id>	-	Berfungsi untuk menghapus riwayat *penetration testing* tertentu berdasarkan pada `scan-id`. Penghapusan ini akan menyebabkan status dan hasil proses hilang.

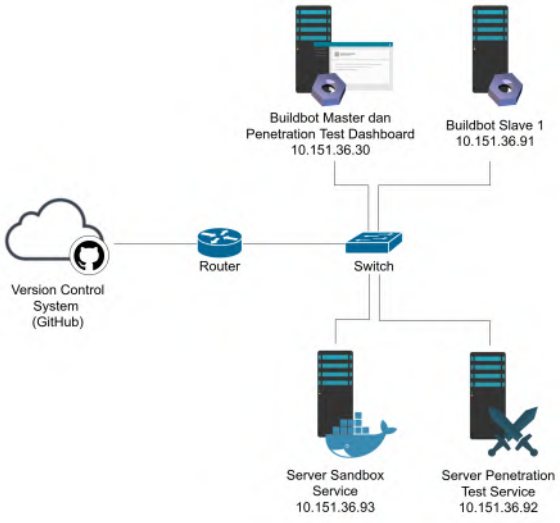
4.5 Rincian Implementasi Buildbot

Buildbot merupakan kerangka kerja untuk aplikasi *Continuous Integration* yang mampu mendeteksi perubahan kode sumber pada VCS, menentukan jadwal *penetration testing* serta memicu aksi *penetration testing*. Implementasi subsistem ini dibagi menjadi beberapa bagian, meliputi implementasi Buildbot Master, implementasi Buildbot Slave 1 dan implementasi Buildbot Slave 2.

BAB 5

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba



Gambar 5.1: Infrastruktur Lingkungan Pengujian

Sesuai dengan diagram infrastruktur yang tertera pada Gambar 5.1, lingkungan pengujian menggunakan empat komponen yang terdiri dari :

- Sebuah komputer sebagai *Penetration Test Dashboard* dan Buildbot Master
  Penetration Test Dashboard berfungsi sebagai sistem yang menjalankan aktivitas *penetration testing* hingga melaporkan hasil pemindaian celah keamanan. Sedangkan Buildbot Master berfungsi sebagai aplikasi pusat kendali *Continuous Integration* sekaligus pengatur jadwal dan pemicu aktivitas *penetration testing* yang akan dilakukan

oleh *Penetration Test Dashboard*.

- Sebuah komputer sebagai Buildbot Slave
  Buildbot Slave berfungsi sebagai pelaksana perintah yang diberikan oleh Buildbot Master.
- Sebuah komputer sebagai *Penetration Test Service*
  Penetration Test Service berfungsi sebagai penyedia alat kakas *penetration testing* yang nantinya akan digunakan oleh *Penetration Test Dashboard*.
- Sebuah komputer sebagai *Sandbox Service*
  Sandbox Service berfungsi sebagai penyedia lingkungan simulasi untuk menjalankan aplikasi yang ingin dipindai dalam sebuah *container* berbasis Docker.

Spesifikasi perangkat keras dan perangkat lunak untuk setiap komponen yang digunakan adalah sebagai berikut:

- *Penetration Test Dashboard* dan Buildbot Master (dalam satu perangkat yang sama):
  - Perangkat Keras:
    * Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
    * RAM 4GB
    * Hard disk 500GB
  - Perangkat Lunak:
    * Sistem operasi Ubuntu LTS 14.04.4
    * NodeJS versi 6.2.1
    * ExpressJS versi 4.13.1
    * ReactJS versi 15.1.0
    * NPM versi 3.9.3
    * Bower versi 1.7.7
    * Nodemon versi 1.9.2
    * MySQL 14.14 Distribusi 5.5.49
    * Python versi 2.7.6
    * Python Virtualenv versi 15.0.1
    * Buildbot versi 0.9.0b8

        * OpenSSH
- Buildbot Slave (dalam virtualisasi Proxmox):
    - Perangkat Keras:
        * Common KVM processor
        * 512MB
        * 34GB QEMU Hard Disk
    - Perangkat Lunak:
        * Sistem operasi Ubuntu LTS 14.04.4
        * Python versi 2.7.6
        * Python Virtualenv versi 15.0.1
        * Buildbot Slave versi 0.9.0b7
        * Git versi 1.9.1
        * OpenSSH
- *Penetration Test Service*:
    - Perangkat Keras:
        * Common KVM processor
        * 993MB
        * 34GB QEMU Hard Disk
    - Perangkat Lunak:
        * Sistem operasi Ubuntu LTS 14.04.4
        * Python versi 2.7.6
        * Python Tornado versi 4.3
        * Docker-py versi 1.8.1
        * GitPython versi 2.0.3
        * Git versi 1.9.1
        * W3af
        * Python w3af-api-client versi 1.1.7
        * Docker versi 1.11.1
        * OpenSSH
- *Sandbox Service*:
    - Perangkat Keras:
        * Common KVM processor
        * 512MB

    * 34GB QEMU Hard Disk
  – Perangkat Lunak:
    * Sistem operasi Ubuntu LTS 14.04.4
    * Python versi 2.7.6
    * Python Tornado versi 4.3
    * Docker-py versi 1.8.1
    * GitPython versi 2.0.3
    * Docker versi 1.11.1
    * Git versi 1.9.1
    * OpenSSH

Konfigurasi alamat IP yang digunakan untuk masing-masing komponen adalah sebagai berikut:

- *Penetration Test Dashboard* menggunakan alamat 10.151.36.30:3000 sebagai layanan *back end* dan alamat 10.151.36.30:4000 sebagai layanan *front end*
- Buildbot Master menggunakan alamat 10.151.36.30:8010
- Buildbot Slave menggunakan alamat 10.151.36.91
- *Penetration Test Service* menggunakan alamat 10.151.36.92:8000 sebagai layanan *web service* dan 10.151.36.92:5000 - 10.151.36.92:5004 sebagai layanan alat kakas *penetration testing*
- *Sandbox Service* menggunakan alamat 10.151.36.93:8000 sebagai layanan *web service* dan 10.151.36.93:9001 hingga seterusnya sebagai alamat *container* aplikasi yang ingin diuji

5.2 Skenario Uji Coba

Pengujian sistem dibagi menjadi dua bagian meliputi uji coba fungsionalitas dan penggunaan sumber daya.

5.2.1 Skenario Uji Coba Fungsionalitas

Uji coba fungsionalitas berfungsi untuk memastikan fitur yang disediakan sistem sudah memenuhi kebutuhan yang tertera dalam diagram kasus penggunaan pada Gambar 3.3. Pengujian dilakukan dengan cara menjalankan fitur-fitur yang telah dibuat dan menguji keberhasilan masing-masing fitur dalam menjalankan fungsinya. Uji coba fungsionalitas meliputi semua kasus penggunaan yang dijelaskan pada Bab 3.2, meliputi:

- Pengujian pengguna dapat melakukan manajemen skenario uji coba.
- Pengujian pengguna dapat melakukan manajemen konfigurasi *sandbox*.
- Pengujian pengguna dapat menjalankan skenario uji coba.
- Pengujian aplikasi *continuous integration* dapat menjalankan skenario uji coba.
- Pengujian pengguna dapat melihat hasil uji coba.
- Pengujian aplikasi *continuous integration* dapat menerima hasil uji coba.
- Pengujian pengguna dapat menerima *e-mail* notifikasi hasil uji coba.

5.2.1.1 Uji Coba Pengguna Mengakses Halaman Skenario

Uji coba ini dilakukan dengan mengakses halaman skenario yang tersedia pada rute `/scenario`. Pengguna akan dihadapkan pada antarmuka utama berupa tabel beserta tombol-tombol aksi untuk membuat skenario baru, menyunting serta menghapus. Semua tombol dicoba dan dipastikan bisa menjalankan fungsinya masing-masing. Skenario lengkap uji coba tertera pada Tabel 5.1.

Tabel 5.1: Prosedur Uji Coba Pengguna Mengakses Halaman Skenario

ID	UJ-001

Tabel 5.1: Prosedur Uji Coba Pengguna Mengakses Halaman Skenario

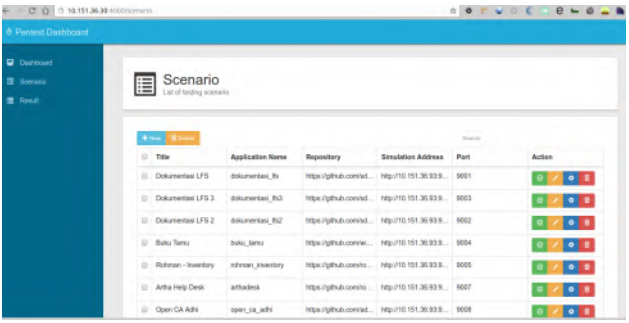
Referensi Use Case	UC-001
Nama	Uji coba pengguna mengakses halaman skenario
Tujuan	Menguji fitur manajemen skenario uji coba yang meliputi daftar skenario, pembuatan skenario baru, penyuntingan serta penghapusan
Kondisi Awal	Pengguna disuguhkan tampilan web sistem
Skenario 1	Pengguna menuju halaman `/scenario` dengan cara memilih menu pada *sidebar*
Masukan	Pilihan menu oleh pengguna
Keluaran	Tampilan tabel berisi daftar skenario yang tersimpan dalam basis data beserta tombol-tombol aksi, seperti Gambar 5.2
Hasil Uji Coba	Berhasil
Skenario 2	Menampilkan formulir isian skenario baru
Masukan	Pengguna menekan tombol `New` untuk menambah skenario baru
Keluaran	Sistem menampilkan formulir isian skenario baru, seperti pada Gambar 5.3
Hasil Uji Coba	Berhasil
Skenario 3	Menyimpan data skenario baru
Masukan	Pengguna mengisi data skenario baru pada formulir lalu menekan tombol `Save` untuk menyimpan
Keluaran	Data tersimpan pada basis data

Tabel 5.1: Prosedur Uji Coba Pengguna Mengakses Halaman Skenario

Hasil Uji Coba	Berhasil
Skenario 4	Menampilkan formulir penyuntingan skenario
Masukan	Pengguna menekan tombol dengan simbol pensil pada salah satu baris tabel
Keluaran	Sistem menampilkan formulir penyuntingan skenario, seperti Gambar 5.4
Hasil Uji Coba	Berhasil
Skenario 5	Menyimpan perubahan terhadap skenario
Masukan	Pengguna menyunting data skenario pada formulir lalu menekan tombol `Save` untuk menyimpan
Keluaran	Data tersimpan pada basis data
Hasil Uji Coba	Berhasil
Skenario 6	Menghapus skenario
Masukan	Pengguna menekan tombol dengan simbol tempat sampah
Keluaran	Data dihapus dari basis data
Hasil Uji Coba	Berhasil

5.2.1.2 Uji Coba Pengguna Mengakses Halaman Konfigurasi Sandbox

Uji coba ini dilakukan dengan cara mengakses halaman pada rute `/scenario/configs/:id` melalui tombol dengan simbol gerigi pada masing-masing baris tabel daftar skenario. Skenario uji coba selengkapnya tertera pada Tabel.



Gambar 5.2: Tabel Daftar Skenario

Tabel 5.2: Prosedur Uji Coba Pengguna Mengakses Halaman Konfigurasi Sandbox

ID	UJ-002
Referensi Use Case	UC-002
Nama	Uji coba pengguna mengakses halaman konfigurasi *sandbox*
Tujuan	Menguji fitur manajemen konfigurasi *sandbox* yang meliputi daftar *file* konfigurasi, pembuatan *file* baru, penyuntingan serta penghapusan
Kondisi Awal	Pengguna berada pada halaman daftar skenario
Skenario 1	Pengguna menuju halaman `/scenario/configs/:id` dengan menekan tombol dengan simbol gerigi pada tabel
Masukan	Pilihan menu oleh pengguna
Keluaran	Tampilan berisi daftar *file* konfigurasi untuk*sandbox* beserta *editor*
Hasil Uji Coba	Berhasil

Tabel 5.2: Prosedur Uji Coba Pengguna Mengakses Halaman Konfigurasi Sandbox

Skenario 2	Menambah *file* konfigurasi baru
Masukan	Pengguna menekan tombol `New Item` untuk menambah *file* konfigurasi baru
Keluaran	Sistem menampilkan formulir isian *file* konfigurasi baru yang dikemas dalam format *editor* kode sumber
Hasil Uji Coba	Berhasil
Skenario 3	Menyimpan *file* konfigurasi baru
Masukan	Pengguna mengisi judul dan konten *file* baru pada *editor* lalu menekan tombol `Save Changes` untuk menyimpan
Keluaran	Data tersimpan pada basis data
Hasil Uji Coba	Berhasil
Skenario 4	Menyunting *file* konfigurasi
Masukan	Pengguna memilih *file* yang ingin disunting
Keluaran	Sistem menampilkan konten *file* pada *editor*
Hasil Uji Coba	Berhasil
Skenario 5	Menyimpan perubahan terhadap skenario
Masukan	Pengguna menyunting konten pada *editor* lalu menekan tombol `Save Changes` untuk menyimpan
Keluaran	Data tersimpan pada basis data
Hasil Uji Coba	Berhasil
Skenario 6	Menghapus *file* konfigurasi
Masukan	Pengguna menekan tombol dengan simbol silang tepat di sebelah kanan nama *file*

Tabel 5.2: Prosedur Uji Coba Pengguna Mengakses Halaman Konfigurasi Sandbox

Keluaran	Data dihapus dari basis data
Hasil Uji Coba	Berhasil

5.2.1.3 Uji Coba Menjalankan Skenario

Uji coba ini dilakukan dengan cara menekan tombol pada tabel daftar skenario untuk menguji fungsionalitas ini dari sisi pengguna. Sedangkan, dari sisi aplikasi *continuous integration* melalui sebuah rute khusus yang bisa diakses menggunakan protokol HTTP. Skenario selengkapnya tertera pada tabel 5.3

Tabel 5.3: Prosedur Uji Coba Menjalankan Skenario

ID	UJ-003
Referensi Use Case	UC-003
Nama	Uji coba menjalankan skenario *sandbox*
Tujuan	Menguji fitur untuk menjalankan skenario *penetration testing*dari sisi penggunan serta sisi aplikasi *continuous integration*
Kondisi Awal	Pengguna berada pada halaman daftar skenario
Skenario 1	Pengguna menjalankan skenario *penetration testing*
Masukan	Pengguna menekan tombol berwarna hijau dengan simbol tombol putar
Keluaran	Tampilan yang menunjukkan aksi sedang diproses, seperti pada Gambar 5.6
Hasil Uji Coba	Berhasil

Tabel 5.3: Prosedur Uji Coba Menjalankan Skenario

Skenario 2	Aplikasi *continuous integration* menjalankan skenario *penetration testing*
Masukan	*Request* dari aplikasi *continuous integration* dengan protokol HTTP dan metode POST melalui rute `/scenario/run/:id`
Keluaran	Status aksi dalam bentuk JSON
Hasil Uji Coba	Berhasil

5.2.1.4 Uji Coba Melihat Hasil

Uji coba ini dilakukan dengan cara mengunjungi halaman hasil yang terdapat pada rute `/result`. Jika hasil diakses melalui aplikasi *continuous integration* maka hasil dikembalikan dalam bentuk JSON.

Tabel 5.4: Prosedur Uji Coba Melihat Hasil

ID	UJ-004
Referensi Use Case	UC-004
Nama	Uji coba melihat hasil
Tujuan	Menguji fitur untuk melihat hasil *penetration testing*
Kondisi Awal	Pengguna disuguhkan tampilan web sistem
Skenario 1	Pengguna menuju halaman `/result` dengan cara memilih menu pada *sidebar*
Masukan	Pilihan menu oleh pengguna
Keluaran	Tampilan berisi *dropdown* pilihan skenario serta daftar hasil *penetration testing* dalam bentuk tabel

Tabel 5.4: Prosedur Uji Coba Melihat Hasil

Hasil Uji Coba	Berhasil
Skenario 2	Menampilkan hasil berdasarkan pilihan skenario
Masukan	Pengguna memilih skenario yang ingin ditampilkan hasilnya
Keluaran	Sistem menampilkan daftar hasil berdasarkan skenario yang dipilih
Hasil Uji Coba	Berhasil
Skenario 3	Melihat detail hasil *penetration testing*
Masukan	Pengguna menekan tombol `Show Result`
Keluaran	Sistem menampilkan detail hasil *penetration testing* dalam bentuk tabel
Hasil Uji Coba	Berhasil

5.2.1.5 Uji Coba Menerima Notifikasi E-mail

Uji coba ini dilakukan dengan cara memeriksa kotak masuk *e-mail* setelah proses *penetration testing* selesai. Jika terdapat email yang berisi tautan menuju halaman hasil, maka fungsi pengiriman *e-mail* berhasil dieksekusi setelah aktivitas *penetration testing* selesai.

Tabel 5.5: Prosedur Uji Coba Menerima Notifikasi E-mail

ID	UJ-005
Referensi Use Case	UC-005
Nama	Uji coba menerima notifkasi *e-mail*

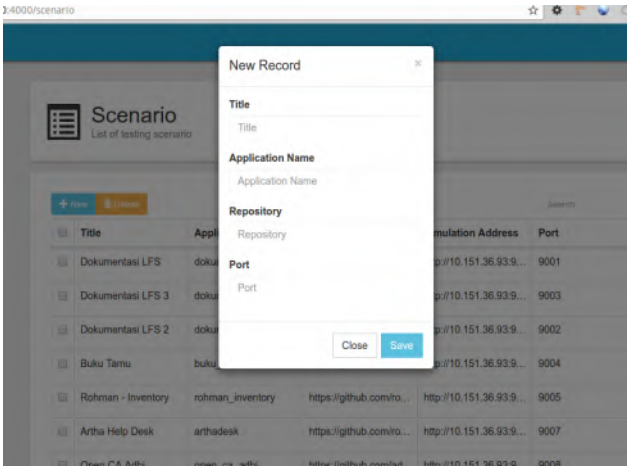
Tabel 5.5: Prosedur Uji Coba Menerima Notifikasi E-mail

Tujuan	Menguji fitur pengiriman notifikasi melalui *e-mail* setelah aktivitas *penetration testing* selesai
Kondisi Awal	Belum ada *e-mail* masuk dari sistem
Skenario 1	Pengguna menerima *e-mail* dari sistem setelah aktivitas *penetration testing* selesai
Masukan	-
Keluaran	*E-mail* masuk berupa pesan bahwa telah terjadi proses *penetration testing* dan tautan menuju halaman hasil *penetration testing*
Hasil Uji Coba	Berhasil

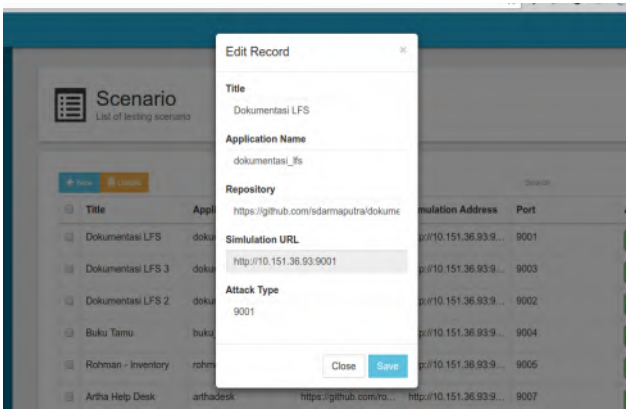
5.2.2 Skenario Uji Coba Penggunaan Sumber Daya

Uji coba penggunaan sumber daya berfungsi untuk mengetahui besar penggunaan memori dan CPU pada saat sistem berjalan. Skenario pengujian dilakukan dengan cara menjalankan aktivitas *penetration testing* terhadap beberapa kode sumber berbeda dalam satu waktu, kemudian mengevaluasi penggunaan memori dan CPU yang terjadi. Skenario uji coba penggunaan sumber daya adalah sebagai berikut:

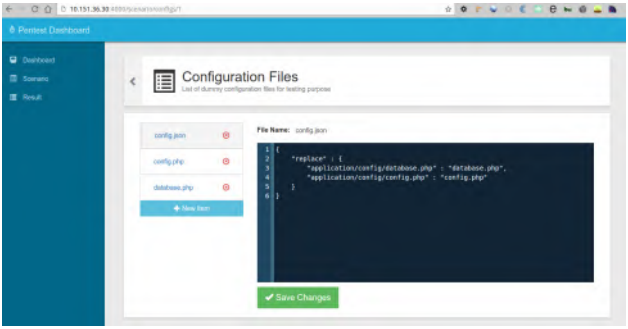
- Menjalankan satu proses *penetration testing*.
- Menjalankan dua proses *penetration testing* terhadap dua kode sumber berbeda dalam waktu yang sama.
- Menjalankan tiga proses *penetration testing* terhadap tiga kode sumber berbeda dalam waktu yang sama.
- Menjalankan empat proses *penetration testing* terhadap empat kode sumber berbeda dalam waktu yang sama.
- Menjalankan lima proses *penetration testing* terhadap lima kode sumber berbeda dalam waktu yang sama.



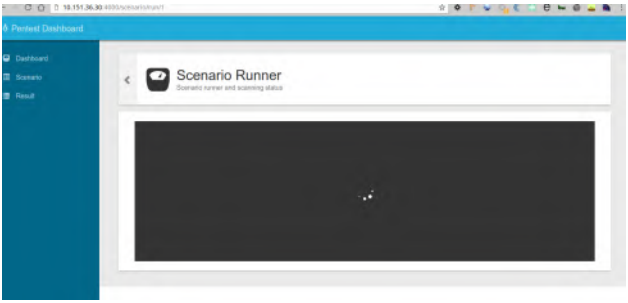
Gambar 5.3: Formulir Tambah Skenario Baru



Gambar 5.4: Formulir Sunting Skenario



Gambar 5.5: Halaman Konfigurasi *Sandbox*



Gambar 5.6: Halaman Pemrosesan Aksi *Penetration Testing*

Masing-masing skenario tersebut dijalankan pada empat kondisi berbeda,antara lain:

5.3 Hasil Uji Coba dan Evaluasi

Berikut ini merupakan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dibuat pada Bab 5.2.

5.3.1 Uji Coba Fungsionalitas

Hasil uji coba fungsionalitas menunjukkan bahwa sistem yang dibuat pada Tugas Akhir ini mampu memenuhi kebutuhan-kebutuhan yang didefinisikan melalui diagram kasus penggunaan pada Bab 3.2. Detail hasil uji coba fungsionalitas tertera pada Tabel 5.6.

Tabel 5.6: Hasil Uji Coba Fungsionalitas

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Hasil Uji Coba**
UC-001	Manajemen skenario uji coba	Terpenuhi
UC-002	Manajemen konfigurasi *sandbox*	Terpenuhi
UC-003	Menjalankan skenario uji coba	Terpenuhi
UC-004	Melihat hasil uji coba	Terpenuhi
UC-005	Menerima *e-mail* notifikasi hasil uji coba	Terpenuhi

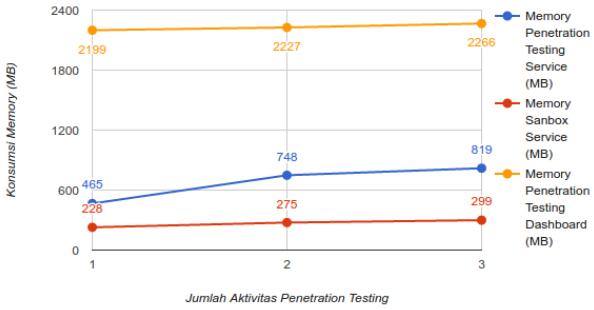
5.3.2 Uji Coba Penggunaan Sumber Daya

Seperti yang sudah dijelaskan pada bab 5.2.2, uji coba penggunaan sumber daya dilakukan bertahap dengan menggunakan jumlah aktivitas *penetration testing*yang berbeda-beda. Pengujian ini akan membandingkan penggunaan memori dan CPU untuk masing-masing skenario. Terdapat empat kondisi berbeda yang digunakan dalam proses uji coba penggunaan sumber daya.

Kondisi pertama adalah ketika *Penetration Test Service* berjalan pada *server* dengan *memory* 1GB dan *processor* dua inti. Hasil uji coba pada kondisi ini tertera pada Gambar 5.7 dan Gambar 5.8. Pada kondisi ini terlihat bahwa *Penetration Test Service* gagal menjalankan fungsinya ketika akan menjalankan empat aktivitas *penetration testing* secara bersamaan. Kegagalan fungsi tersebut juga diikuti dengan terjadinya *hang* pada server, sehingga tidak bisa melanjutkan pada skenario selanjutnya.

Jumlah Aktivitas Penetration Testing	Memory Penetration Testing Service (MB)	Memory Sanbox Service (MB)	Memory Penetration Testing Dashboard (MB)	CPU Pentest Service (%)		CPU Sanbox Service (%)	CPU Pentest Dashboard (%)				Sukses	Gagal
1	465	228	2199	50.53	59.67	43.77	8.40	5.50	4.33	4.57	1	0
2	748	275	2227	57.97	61.10	35.93	6.50	9.13	5.83	4.50	2	0
3	819	299	2266	65.70	68.07	6.87	6.40	8.17	3.93	5.00	3	0
4	-	-	-	-	-	-	-	-	-	-	0	4
5	-	-	-	-	-	-	-	-	-	-	-	-

Gambar 5.7: Hasil Uji Coba dengan Kondisi 1



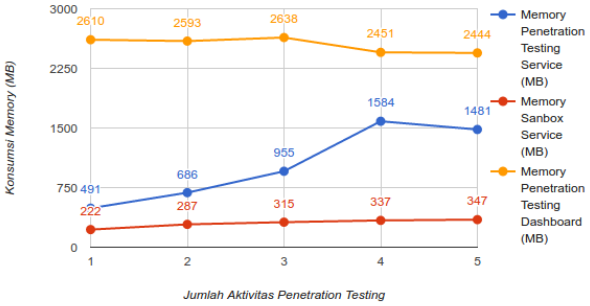
Gambar 5.8: Grafik Hasil Uji Coba dengan Kondisi 1

Kondisi kedua adalah ketika *Penetration Test Service* berjalan pada *server* dengan *memory* 2GB dan *processor* dua inti. Hasil uji coba pada kondisi ini tertera pada Gambar 5.9 dan

Gambar 5.10. Pada kondisi ini terlihat bahwa *Penetration Test Service* gagal menjalankan fungsinya ketika akan menjalankan limat aktivitas *penetration testing* secara bersamaan. Namun, kegagalan fungsi tersebut hanya berlaku untuk satu aktivitas *penetration testing*dan *server* masih bisa berjalan dengan normal.

Jumlah Aktivitas Penetration Testing	Memory Penetration Testing Service (MB)	Memory Sanbox Service (MB)	Memory Penetration Testing Dashboard (MB)	CPU Pentest Service (%)		CPU Sanbox Service (%)	CPU Pentest Dashboard (%)				Sukses	Gagal
1	491	222	2610	41.97	49.87	1.57	7.70	7.77	8.93	5.43	1	0
2	686	287	2593	53.80	53.23	33.47	9.70	9.63	4.87	5.47	2	0
3	955	315	2638	65.13	56.10	33.80	7.00	8.37	6.33	5.47	3	0
4	1584	337	2451	98.07	97.77	38.00	7.70	9.67	7.90	5.67	4	0
5	1481	347	2444	100.00	100.00	86.30	8.10	3.40	3.90	5.30	4	1

Gambar 5.9: Hasil Uji Coba dengan Kondisi 2

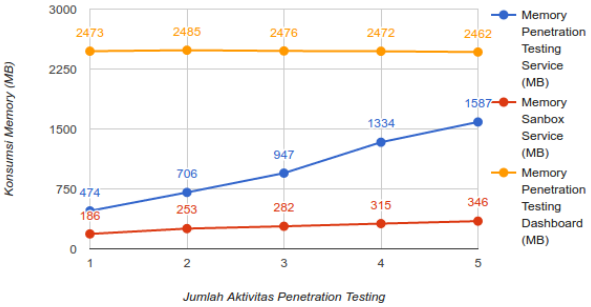


Gambar 5.10: Grafik Hasil Uji Coba dengan Kondisi 2

Kondisi ketiga adalah ketika *Penetration Test Service* berjalan pada *server* dengan *memory* 2GB dan *processor* tiga inti. Hasil uji coba pada kondisi ini tertera pada Gambar 5.11 dan Gambar 5.12. Pada kondisi ini terlihat bahwa *Penetration Test Service* berhasil menjalankan fungsinya hingga lima aktivitas *penetration testing* secara bersamaan.

Jumlah Aktivitas Penetration Testing	Memory Penetration Testing Service (MB)	Memory Sanbox Service (MB)	Memory Penetration Testing Dashboard (MB)	CPU Pentest Service (%)			CPU Sanbox Service (%)	CPU Pentest Dashboard (%)				Sukses	Gagal
1	474	186	2473	35.07	22.07	19.43	32.13	11.17	9.87	7.23	8.07	1	0
2	706	253	2485	31.73	38.07	35.07	21.57	12.63	9.23	14.47	7.80	2	0
3	947	282	2476	59.57	61.37	50.97	38.57	10.63	11.07	12.17	12.53	3	0
4	1334	315	2472	72.17	74.20	77.47	35.70	8.50	5.30	7.87	7.67	4	0
5	1587	346	2462	67.23	89.90	84.70	37.77	9.43	6.20	6.60	6.20	5	0

Gambar 5.11: Hasil Uji Coba dengan Kondisi 3



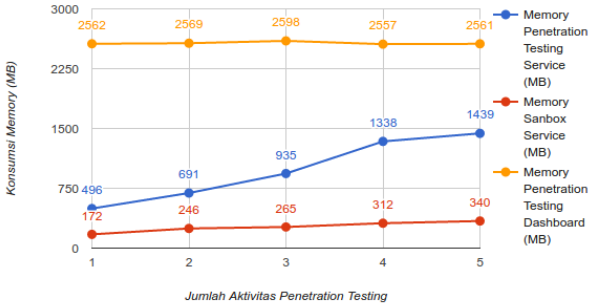
Gambar 5.12: Grafik Hasil Uji Coba dengan Kondisi 3

Kondisi keempat adalah ketika *Penetration Test Service* berjalan pada *server* dengan *memory* 2GB dan *processor* satu inti. Penentuan kondisi ini bertujuan untuk memastikan bahwa jumlah inti *processor* berpengaruh terhadap kemampuan *Penetration Test Service*. Namun, hasil uji coba menunjukkan bahwa sistem mampu menjalankan fungsinya hingga lima aktivitas *penetration testing* secara bersamaan.

Berdasarkan hasil uji coba penggunaan sumber daya, terlihat bahwa kemampuan *Penetration Test Service* ditentukan oleh ketersediaan *memory* pada *server*, sedangkan jumlah inti *processor* tidak serta merta menentukan kemampuan dari sistem ini. Semakin bertambahnya jumlah aktivitas *penetration testing*

Jumlah Aktivitas Penetration Testing	Memory Penetration Testing Service (MB)	Memory Sanbox Service (MB)	Memory Penetration Testing Dashboard (MB)	CPU Pentest Service (%)	CPU Sanbox Service (%)	CPU Pentest Dashboard (%)				Sukses	Gagal
1	496	172	2562	83.77	33.53	20.73	6.90	5.30	7.67	1	0
2	691	246	2569	81.90	30.03	8.40	7.03	7.40	8.87	2	0
3	935	265	2598	98.23	33.27	7.90	9.57	8.43	5.07	3	0
4	1338	312	2557	100.00	21.63	11.20	8.73	6.97	7.03	4	0
5	1439	340	2561	100.00	46.57	12.10	17.13	12.20	11.27	5	0

Gambar 5.13: Hasil Uji Coba dengan Kondisi 4



Gambar 5.14: Grafik Hasil Uji Coba dengan Kondisi 4

yang berjalan dalam waktu yang sama akan menyebabkan kenaikan konsumsi *memory*. Kenaikan konsumi *memory* untuk masing-masing skenario dan kondisi tertera pada Gambar 5.15.

Berdasarkan perhitungan rata-rata kenaikan konsumsi *memory* pada Gambar 5.15, setiap aktivitas *penetration testing* rata-rata membutuhkan *memory* sebesar 261,33MB. Perhitungan ini bisa dijadikan acuan untuk memprediksi besar *memory* yang dibutuhkan untuk menjalankan sejumlah tertentu aktivitas *penetration testing*.

Jumlah Aktivitas Penetration Testing	Kenaikan pada Kondisi 1	Kenaikan pada Kondisi 2	Kenaikan pada Kondisi 3	Kenaikan pada Kondisi 4
1	-	-	-	-
2	263.00	195.00	232.00	195.00
3	71.00	269.00	241.00	244.00
4	-	629.00	387.00	403.00
5	-	-	253.00	101.00
Kenaikan Rata-rata Per Kondisi	**167.00**	**364.33**	**278.25**	**235.75**
Kenaikan Rata-rata Setiap Kondisi	**261.33**			

Gambar 5.15: Kenaikan Konsumsi Memory untuk Masing-masing Skenario dan Kondisi (MB)

(Halaman ini sengaja dikosongkan)

LAMPIRAN A

KODE SUMBER

A.1 Modul W3af Runner

```
import os
import sys
import git
import json
import helper
from shutil import copyfile
from docker import Client

printLog = helper.printLog
protocol = "http://"
agentList = {
    0: {'name': 'w3af0', 'address':
    '10.151.36.92:11000', 'running': 0, 'w3afPort':
    '11000', 'supervisorPort': '9000'},
    1: {'name': 'w3af1', 'address':
    '10.151.36.92:11001', 'running': 0, 'w3afPort':
    '11001', 'supervisorPort': '9001'},
    2: {'name': 'w3af2', 'address':
    '10.151.36.92:11002', 'running': 0, 'w3afPort':
    '11002', 'supervisorPort': '9002'},
    3: {'name': 'w3af3', 'address':
    '10.151.36.92:11003', 'running': 0, 'w3afPort':
    '11003', 'supervisorPort': '9003'},
    4: {'name': 'w3af4', 'address':
    '10.151.36.92:11004', 'running': 0, 'w3afPort':
    '11004', 'supervisorPort': '9004'}
  }

docker_client = Client(base_url = 'unix://var/run/
    docker.sock')

# Create new docker container
def createAgentContainer(app_name, w3af_port,
    supervisor_port):
  printLog("-------Starting Container Creator
    Activities!-------")
  # Create docker container for application sandbox
```

```
25    new_container = docker_client.create_container(
        image = "andresriancho/w3af-api",
26      name = app_name,
27      ports = [5000, 9001],
28      host_config = docker_client.create_host_config(
29        port_bindings = {5000 : w3af_port, 9001:
        supervisor_port}
30        )
31    )
32    printLog("Docker creates:", new_container, " on
        port ", w3af_port)
33    return
34
35  def checkAgentAvailability():
36    containers = docker_client.containers(all=True)
37    for key, agent in agentList.iteritems():
38      if not any(container['Names'][0] == ('/' + agent
        ['name']) for container in containers):
39        createAgentContainer(agent['name'], agent['
        w3afPort'], agent['supervisorPort'])
40    return
41
42  def stopAllAgent():
43    for key, agent in agentList.iteritems():
44      printLog('Stoping agent', agent['name'], '@',
        agent['address'])
45      docker_client.stop(agent['name'])
46      printLog('Removing agent', agent['name'], '@',
        agent['address'])
47      docker_client.remove_container(agent['name'])
48      agent['running'] = 0
49    return
50
51  def stopByAddress(address):
52    counter = 0
53    address = address.replace(protocol, "")
54    for key, agent in agentList.iteritems():
55      if (agent['address'] == address):
56        printLog('Stoping agent', agent['name'], '@',
        agent['address'])
57        docker_client.stop(agent['name'])
```

```
58        printLog('Removing agent', agent['name'], '@',
      agent['address'])
59        docker_client.remove_container(agent['name'])
60        agent['running'] = 0
61        createAgentContainer(agent['name'], agent['
     w3afPort'], agent['supervisorPort'])
62        return {'status': 'success', 'address': agent
     ['address'], 'state': 'stopped'}
63      counter += 1
64    if (counter == len(agentList)):
65      printLog('No agent with specified address!')
66      return {'status': 'failed', 'message': 'No agent
       with specified address!'}
67
68  def startAgent():
69    counter = 0
70    containers = docker_client.containers(all=True)
71    for key, agent in agentList.iteritems():
72      if (agent['running'] == 0):
73        printLog('Starting agent', agent['name'], '@',
      agent['address'])
74        if not any(container['Names'][0] == ('/' +
     agent['name']) for container in containers):
75          createAgentContainer(agent['name'], agent['
     w3afPort'], agent['supervisorPort'])
76        docker_client.start(agent['name'])
77        agent['running'] = 1
78        printLog('Successfully running agent', agent['
     name'], '@', agent['address'])
79        return {'status': 'success', 'address':
     protocol+agent['address']}
80      counter += 1
81    if (counter == len(agentList)):
82      printLog('All agent in use. Can not start new
     task!')
83      return {'status': 'failed', 'message': 'All
     agent in use. Can not start new task!'}
84
85  def main():
86    print "Please use this program as plug-in. Thanks
      !"
```

```
87
88 if __name__ == "__main__":
89   main()
90
91
```

Kode Sumber A.1: Modul W3af Runner pada Penetration Testing Service

A.2 Modul Scanner

```
1  import os
2  import time
3  from w3af_api_client import Connection, Scan
4  import helper
5  from pprint import pprint
6  import threading
7  import requests
8  import json
9  import agentRunner
10
11 printLog = helper.printLog
12 parent_dir = '/home/jiwa/tugasakhir/'
13
14 class ResultRetriever(threading.Thread):
15   def __init__(self, scenarioId, applicationName,
       runningToken, conn, scannerUrl, targetUrl, scan):
16     printLog("Thread for", applicationName, "created
       ")
17     threading.Thread.__init__(self)
18     self.scenarioId = scenarioId
19     self.name = applicationName
20     self.runningToken = runningToken
21     self.conn = conn
22     self.targetUrl = targetUrl
23     self.scannerUrl = scannerUrl
24     self.scan = scan
25
26   def run(self):
27     postUrl = "http://10.151.36.30:3000/runner/
       storeResult/"+self.scenarioId
```

```
28      printLog("Running thread task")
29      results = retrieveResults(self.conn, self.
        targetUrl, self.scan)
30
31      payloads = {
32        'applicationName': self.name,
33        'scenarioId': self.scenarioId,
34        'targetUrl': self.targetUrl,
35        'scannerUrl': self.scannerUrl,
36        'runningToken': self.runningToken,
37        'results': json.dumps(results['results'])
38      }
39
40      req = requests.post(postUrl, data=payloads)
41      printLog(req.text)
42      printLog(req.status_code)
43
44      res = agentRunner.stopByAddress(self.scannerUrl)
45
46      printLog(res)
47      printLog("thread finished")
48
49  class Scanner(object):
50    def __init__(self):
51      printLog("Scanner object created")
52
53    def initConnection(self, scannerUrl):
54      printLog("Initialize connection with scanner at
        ", scannerUrl)
55
56      while True:
57        try:
58          printLog("Trying initialization for scanner
        :", scannerUrl)
59          conn = Connection(scannerUrl)
60          ver = conn.get_version()
61          if (ver is not None):
62            printLog("Version: ", conn.get_version())
63            printLog("Scanner initialized: ",
        scannerUrl)
64            break
```

```
65          else:
66            pass
67        except Exception, e:
68          pass
69        else:
70          pass
71        finally:
72          pass
73        time.sleep(3)
74
75      return conn
76
77    def startScanner(self, scenarioId, applicationName
      , runningToken, conn, scannerUrl, targetUrl):
78      scanProfilePath = os.path.join(parent_dir, "w3af
      /profiles/OWASP_TOP10.pw3af")
79      scanProfile = file(scanProfilePath).read()
80      targetUrls = [targetUrl]
81
82      printLog("Starting scanner for target ",
      targetUrl)
83      scan = Scan(conn)
84      try:
85        scan.start(scanProfile, targetUrls)
86      except Exception, e:
87        printLog("Error while starting scanner for
      target", targetUrl, ":", e)
88      else:
89        pass
90      finally:
91        pass
92
93      printLog("Checking status for target", targetUrl
      )
94      while True:
95        try:
96          stat = scan.get_status()
97          printLog("Stat for target", targetUrl, ":",
      stat)
98          if stat is not None:
99            break
```

```
100
101         except Exception, e:
102           printLog("Stat for target", targetUrl, ":",
      e)
103           pass
104         time.sleep(3)
105
106
107       # Check if scanner has running
108       printLog("Checking running status for",
      targetUrl)
109       while True:
110         try:
111           status = scan.get_status()
112           printLog("Running Status:", status['
      is_running'], "for target:", targetUrl)
113           if (status['is_running'] == True):
114             printLog("scan started for target: ",
      targetUrl)
115             break
116           else:
117             pass
118         except Exception as e:
119           printLog("Target ", targetUrl, "error
      occured: ", e)
120           pass
121         else:
122           pass
123         finally:
124           pass
125         time.sleep(3)
126
127       printLog("starting thread for", targetUrl)
128       retrieverThread = ResultRetriever(scenarioId,
      applicationName, runningToken, conn, scannerUrl,
      targetUrl, scan)
129
130       # Check if thread has been started
131       while True:
132         try:
```

```
133         printLog("trying to start thread for",
    targetUrl)
134         retrieverThread.start()
135         printLog("thread for target", targetUrl, "
    live status: ", retrieverThread.isAlive())
136         if not retrieverThread.isAlive():
137           pass
138         else:
139           break;
140       except Exception, e:
141         pass
142       else:
143         pass
144       finally:
145         pass
146       time.sleep(3)
147
148
149     return {'status': "success", 'state': "running",
     'token': runningToken, 'scenarioId':scenarioId,
    'applicationName': applicationName}
150
151   def initTask(self, scenarioId, applicationName,
     runningToken, scannerUrl, targetUrl):
152     conn = self.initConnection(scannerUrl)
153     result = self.startScanner(scenarioId,
    applicationName, runningToken, conn, scannerUrl,
    targetUrl)
154     return result
155
156
157 def retrieveResults(conn, targetUrl, scan):
158   printLog("Start retrieving results for target ",
     targetUrl)
159   while True:
160     try:
161       status = scan.get_status()
162       printLog("Running Status:", status['is_running
    '], "for target:", targetUrl)
163       if (status['is_running'] == False):
164         break
```

```
165     except Exception as e:
166       printLog("Target ", targetUrl, "error occured:
        ", e)
167       pass
168     time.sleep(3)
169
170   summary = scan.get_findings()
171   finalResult = {'results': {}}
172
173   for index in range(len(summary)):
174     currentSummary = summary[index].resource_data
175     tmp_result = { index: {
176         'targetUrl': currentSummary['url'],
177         'vulnerability': currentSummary['name'],
178         'longDescription': currentSummary['
      long_description'],
179         'description': currentSummary['desc'],
180         'severity': currentSummary['severity'],
181         'recommendation': currentSummary['
      fix_guidance'],
182         'references': currentSummary['references']
183       }}
184     if 'urls' in currentSummary:
185       tmp_result['targetUrls'] = currentSummary['
      urls']
186     finalResult['results'].update(tmp_result)
187   return finalResult
188
189 def main():
190   print "Please use this program as plug-in. Thanks
      !"
191
192 if __name__ == "__main__":
193   main()
194
195
196
197
```

Kode Sumber A.2: Modul Scanner pada Penetration Testing Service

A.3 Penetration Testing Web Service

```
import tornado.ioloop
import tornado.web
from tornado import gen
from cStringIO import StringIO
from multiprocessing.pool import ThreadPool
import sys
import helper
import agentRunner
from scanner import Scanner

printLog = helper.printLog
_workers = ThreadPool(10)

def runBackground(function, callback, args=(), kwds
    ={}):
  def _callback(result):
    tornado.ioloop.IOLoop.instance().add_callback(
    lambda: callback(result))
  _workers.apply_async(function, args, kwds,
    _callback)

class Runner(tornado.web.RequestHandler):
  def get(self, *args, **argv):
    self.set_header("Access-Control-Allow-Origin",
    "*")
    self.set_header("Access-Control-Allow-Headers",
    "x-requested-with")
    self.set_header("Access-Control-Allow-Methods",
    "POST, GET")
    if (args[0] == None):
      res = agentRunner.startAgent()
      self.write(res)
    else:
      self.write(args[0])

  def delete(self, *args, **argv):
    self.set_header("Access-Control-Allow-Origin",
    "*")
```

```
32      self.set_header("Access-Control-Allow-Headers",
        "x-requested-with")
33      self.set_header("Access-Control-Allow-Methods",
        "POST, GET")
34      if (args[0] != None):
35        res = agentRunner.stopByAddress(args[0])
36        self.write(res)
37
38  class ScannerAgent(tornado.web.RequestHandler):
39    @tornado.web.asynchronous
40    def post(self, *args, **argv):
41      scenarioId = self.get_argument("scenario_id")
42      applicationName = self.get_argument("
        application_name")
43      runningToken = self.get_argument("running_token
        ")
44      targetUrl = self.get_argument("target_url")
45      scannerUrl = self.get_argument("scanner_url")
46
47      currScanner = Scanner()
48      runBackground(currScanner.initTask, self.
        on_complete, (scenarioId, applicationName,
        runningToken, scannerUrl, targetUrl))
49
50    def on_complete(self, res):
51      self.set_header("Access-Control-Allow-Origin",
        "*")
52      self.set_header("Access-Control-Allow-Headers",
        "x-requested-with")
53      self.set_header("Access-Control-Allow-Methods",
        "POST, GET")
54      self.write(res)
55      self.finish()
56
57  def make_app():
58    printLog("Application started")
59    return tornado.web.Application([
60      (r"/run/?([0-9.:]+)?", Runner),
61      (r"/scan/?([0-9.:]+)?", ScannerAgent)
62    ])
63
```

```
64 if __name__ == "__main__":
65   port = 8000
66   agentRunner.checkAgentAvailability()
67   agentRunner.stopAllAgent()
68   agentRunner.checkAgentAvailability()
69   application = make_app()
70   application.listen(port)
71   printLog('Listening on port ' + str(port))
72   tornado.ioloop.IOLoop.current().start()
73
74
75
76
```

Kode Sumber A.3: Web Service untuk Penetration Testing Service

A.4 Modul Git Puller

```
1  import os
2  import sys
3  import git
4  import json
5  import base64
6  import helper
7  import pprint
8  from shutil import copyfile
9  from docker import Client
10
11 printLog = helper.printLog
12
13 docker_client = Client(base_url = 'unix://var/run/
       docker.sock')
14 parent_dir = '/home/jiwa/tugasakhir/app-sandbox/'
15
16 class Puller(object):
17   def __init__(self):
18     printLog("Puller object created")
19
20   # Create sanbox directory
```

```
21  def setupDirectory(self, app_dir, www_dir):
22    printLog("-------Starting Directory Operations
      !-------")
23    printLog('Checking directory ', app_dir)
24    if not os.path.exists(app_dir):
25      printLog('Directory ', app_dir, " not exists.
      Creating it...")
26      os.mkdir(app_dir)
27    else:
28      printLog('Directory ', app_dir, " already
      exists.")
29    return
30
31  # Create configuration files
32  def createConfigurations(self, app_dir,
      config_file, custom_files):
33    printLog("-------Starting Configuartion File
      Setup!-------")
34
35    try:
36      customFiles = json.loads(custom_files)
37      for (key, item) in customFiles.items():
38        customFilePath = os.path.join(app_dir, item
      ["title"])
39        customFileContent = base64.b64decode(item["
      content"])
40        fcustom = open(customFilePath, 'w+')
41        printLog("Writing to ", item["title"])
42        fcustom.write(customFileContent)
43        fcustom.close()
44    except ValueError, e:
45      printLog("Cannot load custom files: ", e)
46      return
47
48    return
49
50
51  # Pull or clone application from VCS based on
      availability at local server
52  def fetchFromRepo(self, repo_url, www_dir):
```

```
53      printLog("-------Starting Git Activities
        !-------")
54      if os.path.exists(www_dir):
55        printLog("Application directory already exists
        !")
56        printLog("Pulling from repository...")
57        repo = git.cmd.Git(www_dir)
58        repo.pull() # pull if directory exists
59      else:
60        printLog("Cloning repository...")
61        git.Repo.clone_from(repo_url, www_dir) # clone
         if directory not exists yet
62      return
63
64    # Replace application's configuration files for
        production-like environment
65    def setupConfigurations(self, app_dir, www_dir):
66      printLog("-------Starting Configuration
        Activities!-------")
67      # Get values from config.json
68      configFilePath = os.path.join(app_dir, 'config.
        json')
69      printLog("Looking for ", configFilePath)
70      if (os.path.exists(configFilePath)):
71        printLog("Configuration file found at ",
        configFilePath)
72        with open(configFilePath) as data_file:
73          try:
74            configurations = json.load(data_file)
75          except ValueError, e:
76            printLog("Cannot load configuration: ", e)
77            return
78
79        # Replace configurations files
80        for (target_file, config_file) in
        configurations["replace"].items():
81          if os.path.exists(os.path.join(app_dir,
        config_file)):
82            printLog("replacing", target_file, "with",
         config_file)
```

```
83            copyfile(os.path.join(app_dir, config_file
    ), os.path.join(www_dir, target_file))
84          else:
85            printLog("File not found:", os.path.join(
    app_dir, config_file))
86        return
87      else:
88        printLog("File not found: config.json")
89        return
90
91    # Create new docker container
92    def createContainer(self, app_name, www_dir,
    app_port):
93      printLog("-------Starting Container Creator
    Activities!-------")
94      # Create docker container for application
    sandbox
95      new_container = docker_client.create_container(
    image = "richarvey/nginx-php-fpm",
96        name = app_name,
97        ports = [80],
98        host_config = docker_client.create_host_config
    (
99          port_bindings = {80 : app_port},
100           binds = [www_dir + ":/usr/share/nginx/html
    "]
101         )
102     )
103     printLog("Docker creates:", new_container, " on
    port ", app_port)
104     return
105
106   # Set up sandbox for application virtualization
107   def setupSandbox(self, app_name, www_dir, app_port
    ):
108     printLog("-------Starting Container Activities
    !-------")
109     containers = docker_client.containers(all=True)
110
111     # Check if container name already exists
```

```
112     if any(container['Names'][0] == ('/' + app_name)
        for container in containers):
113       printLog("Container already running!")
114       printLog("Stopping container:", app_name)
115       docker_client.stop(app_name)
116       printLog("Removing container:", app_name)
117       docker_client.remove_container(app_name)
118       printLog("Recreating container:", app_name)
119       self.createContainer(app_name, www_dir,
      app_port)
120     else:
121       printLog("No container available for those
      informations!")
122       printLog("Creating container:", app_name)
123       self.createContainer(app_name, www_dir,
      app_port)
124
125     # Start application container
126     docker_client.start(app_name)
127     printLog("Starting container:", app_name)
128     return
129
130   # Initialize task
131   def initTask(self, app_name, repo_url, app_port,
      config_file, custom_files):
132     # Terminate program when no arguments presented
133     if (app_name is None) or (repo_url is None):
134       printLog("Please provide application name and
      repository URL!")
135       sys.exit()
136     else:
137       www_dir = os.path.join(parent_dir, app_name, '
      www')
138       app_dir = os.path.join(parent_dir, app_name)
139
140       printLog("Initializing task...")
141       printLog("Application name:", app_name)
142       printLog("Repository URL:", repo_url)
143       printLog("Application directory:", www_dir)
144
145       self.setupDirectory(app_dir, www_dir)
```

```
146         self.createConfigurations(app_dir, config_file
    , custom_files)
147         self.fetchFromRepo(repo_url, www_dir)
148         self.setupConfigurations(app_dir, www_dir)
149         self.setupSandbox(app_name, www_dir, app_port)
150 
151 def main():
152   argv = sys.argv
153   if (len(argv) < 2):
154     printLog("Program terminated!")
155     printLog("Usage: puller.py [application name] [
      repository url] [application port]")
156     sys.exit()
157   else:
158     # Set up initial variables
159     app_name = argv[1]
160     repo_url = argv[2]
161     app_port = argv[3]
162 
163     puller = Puller()
164     puller.initTask(app_name, repo_url, app_port)
165 
166 if __name__ == "__main__":
167   main()
168 
169 
```

Kode Sumber A.4: Modul Git Puller pada Sandbox Service

A.5 Sandbox Web Service

```
1 import tornado.ioloop
2 import tornado.web
3 from tornado import gen
4 from cStringIO import StringIO
5 from multiprocessing.pool import ThreadPool
6 import sys
7 from puller import Puller
8 import helper
```

```
9
10 printLog = helper.printLog
11 serverUrl = "http://10.151.36.93"
12 runningTask = []
13
14 _workers = ThreadPool(10)
15
16 def runBackground(function, callback, args=(), kwds
       ={}):
17   def _callback(result):
18     tornado.ioloop.IOLoop.instance().add_callback(
       lambda: callback(result))
19   _workers.apply_async(function, args, kwds,
       _callback)
20
21 def getTaskList(memory_file):
22   taskList = []
23   with open(memory_file) as memory:
24     for line in memory:
25       taskList.append(line.replace('\n', ''))
26     memory.close()
27   return taskList
28
29
30 def rewriteTaskList(memory_file, items=[]):
31   memory = open(memory_file, 'w')
32   for item in items:
33     memory.write(item + '\n')
34   memory.close()
35   return
36
37 def startDeploy(appName, repoName, appPort,
       configFile, customFiles):
38   taskList = getTaskList('runningTask.conf')
39
40   if appName not in taskList:
41     printLog('Task for', appName, 'started!')
42     taskList = getTaskList('runningTask.conf')
43     taskList.append(appName)
44     rewriteTaskList('runningTask.conf', taskList)
45     puller = Puller()
```

```
46      puller.initTask(appName, repoName, appPort,
        configFile, customFiles)
47      return {'state': 'finished', 'status': 'success
        ', 'app_name': appName, 'app_port': appPort, '
        app_server': serverUrl}
48    else:
49      return {'state': 'running', 'status': 'running
        ','app_name': appName, 'app_port': appPort, '
        app_server': serverUrl}
50
51
52  class PullAndDeploy(tornado.web.RequestHandler):
53    def get(self):
54      self.write('Nothing to do.')
55
56    @tornado.web.asynchronous
57    def post(self):
58      appName = self.get_argument('app_name')
59      repoName = self.get_argument('repo_name')
60      appPort = self.get_argument('app_port')
61      configFile = self.get_argument('config_file',
        default=None)
62      customFiles = self.get_argument('custom_files',
        default=None)
63
64      runBackground(startDeploy, self.on_complete, (
        appName, repoName, appPort, configFile,
        customFiles))
65
66
67    def on_complete(self, res):
68      if (res.get('status') == 'success'):
69        taskList = getTaskList('runningTask.conf')
70        taskList.remove(res.get('app_name'))
71        rewriteTaskList('runningTask.conf', taskList)
72
73      self.set_header("Access-Control-Allow-Origin",
        "*")
74      self.set_header("Access-Control-Allow-Headers",
        "x-requested-with")
```

```
75      self.set_header("Access-Control-Allow-Methods",
        "POST, GET")
76      self.write(res)
77      self.finish()
78
79
80  def make_app():
81    printLog("Application started")
82    return tornado.web.Application([
83      (r"/deploy", PullAndDeploy)
84    ])
85
86  if __name__ == "__main__":
87    port = 8000
88    rewriteTaskList('runningTask.conf', [])
89    application = make_app()
90    application.listen(port)
91    printLog('Listening on port ' + str(port))
92    tornado.ioloop.IOLoop.current().start()
93
94
95
```

Kode Sumber A.5: Web Service untuk Sandbox Service

A.6 Modul Helper

```
1  import datetime
2
3  def printLog(*string):
4    print datetime.datetime.now().strftime("%Y-%m-%d %
       H:%M:%S"), '->',
5    for i in string:
6      print i,
7    print ''
8
9  def main():
10   print "Helper library!"
11
```

```
12 if __name__ == "__main__":
13   main()
14
15
```

Kode Sumber A.6: Modul Helper

A.7 Modul Client

```
1  import requests
2  import json
3
4
5  class Client(object):
6    def __init__(self, primaryUrl, scenarioId):
7      self.primaryUrl = primaryUrl
8      self.scenarioId = scenarioId
9      self.runnerUrl = self.primaryUrl + "/runner/run
       /" + str(self.scenarioId)
10
11   def runScanner(self):
12     req = requests.post(self.runnerUrl)
13     runningStatus = json.loads(req.text)
14     self.runningStatus = runningStatus
15     self.resultUrl = self.primaryUrl + "/result/
       scenario/" + str(self.scenarioId) + "/" + self.
       runningStatus['token']
16     return runningStatus
17
18   def getRunningStatus(self):
19     return self.runningStatus
20
21   def getResults(self):
22     req = requests.get(self.resultUrl)
23
24     if (req.text != "null"):
25       resultData = json.loads(req.text)
26       resultDetail = json.loads(resultData['detail
       '])
```

```
27        self.resultDetail = resultDetail
28
29        return resultDetail
30      else:
31        self.resultDetail = []
32        return req.text
33
34
```

Kode Sumber A.7: Modul Client

A.8 Modul Penetration Testing Client

```
1  import sys
2  import time
3  from pentester_client import Client
4
5  def execute(primaryUrl, scenarioId):
6    client = Client(primaryUrl, scenarioId)
7
8    print "Connecting to pentest dashboard"
9    res = client.runScanner()
10   print res
11
12   print "Getting running status"
13   stat = client.getRunningStatus()
14   print stat
15
16   print "Getting results"
17   while True:
18     result = client.getResults()
19     if (result != "null"):
20       print result
21       break
22     time.sleep(3)
23
24 def main():
25   argv = sys.argv
26   if (len(argv) < 2):
```

```
27      print "Program terminated!"
28      print "Usage: client.py [dashboard url] [
        scenario ID]"
29      sys.exit()
30    else:
31      # Set up initial variables
32      primaryUrl = argv[1]
33      scenarioId = argv[2]
34
35      execute(primaryUrl, scenarioId)
36
37  if __name__ == "__main__":
38    main()
39
40
```

Kode Sumber A.8: Modul Penetration Testing Client

(Halaman ini sengaja dikosongkan)

BAB 6

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba serta evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Berdasarkan proses perancangan, implementasi dan pengujian terhadap sistem pada Tugas Akhir ini, dapat diambil beberapa kesimpulan sebagai berikut:

1. Sistem yang dirancang pada Tugas Akhir ini mampu membuktikan bahwa aktivitas *penetration testing* bisa dijalankan pada aplikasi *continuous integration* dan mendukung proses evaluasi terhadap kode sumber suatu aplikasi.
2. Salah satu cara menggabungkan aktivitas *penetration testing* dengan aplikasi *continuous integration* adalah dengan menjadikan sistem *penetration testing* sebagai sebuah layanan pihak ketiga yang menyediakan antar muka API bagi aplikasi *continuous integration*.
3. Aspek yang perlu diperhatikan jika ingin menggabungkan aktivitas *penetration testing* dengan aplikasi *continuous integration* adalah sumber daya perangkat keras, terutama memori sebab alat kakas *penetration testing* mengonsumsi memori cukup banyak dalam sekali beraktivitas.
4. Sumber daya yang paling mempengaruhi fungsionalitas sistem adalah ketersediaan *memory* pada *server Penetration Test Service*, sebab sistem ini mengonsumsi *memory* dalam jumlah besar untuk setiap aktivitas *penetration testing* yang dijalankan, yakni sebesar rata-rata 261.33MB.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Sistem *penetration testing* yang dirancang pada Tugas Akhir ini masih memiliki kekurangan dalam pengelolaan sumber daya, terutama memori, sehingga diperlukan penelitian lebih lanjut mengenai cara menghemat konsumsi memori.
- Alat kakas *penetration testing* yang digunakan pada sistem ini masih terbatas pada W3af, sehingga perlu dikembangkan lagi agar sistem mampu menjalankan alat kakas *penetration testing* lainnya.

DAFTAR PUSTAKA

[1] Pulei Xiong and Lia Peyton **An Automated Method of Penetration Testing in Computing, Communications and IT Applications**, Conference (ComComAp), 2014 IEEE, 2014, pp. 211 - 216, DOI:10.1109/ComComAp.2014.7017198.

[2] Pulei Xiong and Lia Peyton **A Model-Driven Penetration Test Framework for Web Applications**, 2010 Eight Annual International Conference on Privacy,Security and Trust, pp. 173 - 180, DOI: 10.1109/PST.2010.5593250.

[3] Martin Fowler, **Continuous Integration**, [Online], `http://martinfowler.com/articles/continuousIntegration.html`, diakses tanggal 01 Desember 2015

[4] Software Freedom Conservancy, Inc, **About Version Control**, [Online], `https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`, diakses tanggal 01 Desember 2015

[5] Joyent, Inc, **About Node.js**, [Online], `https://nodejs.org/en/about/`, diakses tanggal 01 Desember 2015

[6] OWASP Foundation, **Web Application Penetration Testing**, [Online], `https://www.owasp.org/index.php/Web_Application_Penetration_Testing`, diakses tanggal 01 Desember 2015

[7] Software Freedom Conservancy, Inc, **About Buildbot**, [Online], `http://buildbot.net/about.html`, diakses tanggal 01 Desember 2015

[8] PHP Group, **About PHP**, [Online], `http://www.php.net/`, diakses tanggal 01 Desember 2015

[9] OWASP Foundation, **About SQL Injection**, [Online], https://www.owasp.org/index.php/SQL_Injection, diakses tanggal 01 Desember 2015

[10] OWASP Foundation, **About Cross Site Scripting**, [Online], https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), diakses tanggal 01 Desember 2015

[11] OWASP Foundation, **About Directory Indexing**, [Online], https://www.owasp.org/index.php/OWASP_Periodic_Table_of_Vulnerabilities_-_Directory_Indexing, diakses tanggal 01 Desember 2015

[12] OWASP Foundation, **About Path Traversal**, [Online], https://www.owasp.org/index.php/Path_Traversal, diakses tanggal 01 Desember 2015

[13] Robert Auger, **About Remote File Inclusion**, [Online], http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion, diakses tanggal 01 Desember 2015

[14] W3af Community, **W3af Official Documentation**, [Online], http://docs.w3af.org/en/latest/, diakses tanggal 10 Januari 2016

[15] Michael Trier, **GitPython Documentation**, [Online], http://gitpython.readthedocs.io/, diakses tanggal 10 Januari 2016

[16] Aanand Prasad, **Docker-py Documentation**, [Online], http://docker-py.readthedocs.io/, diakses tanggal 25 Januari 2016

[17] Facebook, Inc, **React JS Documentation**, [Online], https://facebook.github.io/react/, diakses tanggal 15 Februari 2016

BIODATA PENULIS

I Gede Putu Surya Darma Putra, lahir pada tanggal 21 September 1994 di Denpasar, Bali sebagai putra pertama dari pasangan I Wayan Sudiarta dan Ni Luh Sumarni. Penulis adalah orang yang cenderung pendiam dan pemikir. Ketertarikan penulis terhadap dunia Teknologi Informasi dimulai dari bangku SMP ketika salah seorang temannya memperkenalkan penulis dengan mesin pencari Google. Ketertarikan itu pun berlanjut dan membawa penulis menelusuri dunia *online game* dan *chat messenger* yang pada masa itu sedang naik daun. Sejak masa itulah penulis mulai menggeluti dunia Teknologi Informasi yang diawali dengan mengasah kemampuan menggunakan aplikasi perkantoran dan *editor* gambar. Perlahan penulis mulai terjun ke dalam dunia pemrograman komputer yang akhirnya mengantarnya menjadi seorang mahasiswa jurusan Teknik Informatika salah satu institut teknologi ternama di Indonesia, Institut Teknologi Sepuluh Nopember. Hingga buku ini dikeluarkan, penulis fokus mengasah kemampuan dalam pengembangan *software* serta keamanan aplikasi berbasis web. Sebagai pengembang *software* penulis mendalami ilmu terkait *user experience* dan *user interface* baik dalam aplikasi berbasis web maupun perangkat *mobile*. Selain menggeluti dunia Teknologi Informasi penulis juga sedang mengasah kemampuan menulis dan desain grafis. Pemikiran-pemikiran penulis dituangkan dalam blog pribadinya yang bisa diakses melalui alamat `http://yasurya.com`. Di samping itu, penulis merupakan penikmat seni, baik itu seni rupa, musik, tari maupun teater serta dunia kuliner dan bercocok tanam. Penulis memiliki cita-cita ingin menikmati hari tua sebagai seseorang yang memiliki keseharian pergi ke ladang. Oleh karena itu, selama masa mudanya penulis bekerja keras mengumpulkan modal dan ingin menghabiskan waktu untuk mengeksplorasi dunia

Teknologi Informasi. Penulis terbuka dengan ajakan diskusi, baik mengenai Tugas Akhir ini maupun hal lain dan bisa dihubungi melalui surel surya.igede@gmail.com.